UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA

INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

DISEÑO DE INTERFAZ USB CON PLATAFORMAS FPGAS

Autor: Maite González Sánchez

Tutor: Marta Portela García

Codirector: Mario García Valderas

Resumen

La mayoría de las plataformas FPGA (*Field-programmable gate array*) disponen de una interfaz USB para comunicar el hardware implementado con el PC. Para ello utilizan un microprocesador que controla la interfaz y que requiere el desarrollo del software para su funcionamiento. El objetivo de este proyecto es prescindir de dicho microprocesador, diseñando una interfaz hardware que pueda usarse en cualquier FPGA. Para la implementación de la interfaz USB, se utiliza el controlador USB FT120, que se conecta a la FPGA a través de un bus paralelo genérico de 8 bits.

Tras estudiar en detenimiento el protocolo USB, se considera como objetivo suficiente para este proyecto la implementación del proceso de enumeración del dispositivo USB. Este consiste en la inicialización y el establecimiento de la conexión entre el PC y el dispositivo FT120. El sistema implementado en la FPGA se encargará del control de la comunicación, gestionando todo lo necesario para el establecimiento de la conexión USB.

El desarrollo del sistema se ha llevado a cabo mediante el lenguaje de descripción hardware VHDL (*VHSIC Hardware Description Language*), y se ha utilizado el simulador ModelSim para editar, compilar, simular y depurar el diseño realizado. Una vez comprobado su funcionamiento teórico, se procede a su implementación hardware en la FPGA Spartan 3E de Xilinx. Para ello se utiliza la herramienta Xilinx-ISE (*Integrated Software Environment*), que permite sintetizar y verificar el diseño, implementarlo, crear los datos de configuración y por último configurar la FPGA a partir del fichero bitstream resultante.

Palabras clave: FPGA, interfaz USB, FT120, protocolo USB, VHDL, ModelSim, Xilinx-ISE.

Abstract

The majority of the FPGA (*Field-programmable gate array*) platforms have a USB interface to communicate the implemented hardware with the PC. These FPGA use a microprocessor that controls the interface and requires the software development for its correct operation. The goal of this project is the development of a substitute for this microprocessor, designing a hardware interface that can be used in any FPGA. The FT120 USB controller is used to implement the USB interface, which is connected to the FPGA through a generic 8-bit parallel bus.

After having studied in detail the USB protocol, the enumeration process implementation of USB device is considered a sufficient objective for this project. This process consists in the initialization and establishment of the connection between the PC and the FT120 device. The system implemented by the FPGA is responsible of the communication control, managing all the necessary elements to establish the USB connection.

The system development has been performed using VHDL (*VHSIC Hardware Description Language*), a hardware description language, and the ModelSim simulator, which was used to edit, compile, simulate and debug the realized design. After checking its theoretical operation, its hardware implementation was done in the Xilinx FPGA Spartan 3E. The Xilinx-ISE (*Integrated Software Environment*) tool is used for this purpose, which allows: synthesize and check the design, implement the design, create the configuration data and configure the FPGA from the resulting bitstream file.

Keywords: FPGA, USB interface, FT120, USB protocol, VHDL, ModelSim, Xilinx-ISE.

Índice

Índice de Figuras

Índice de Tablas

1. Introducción

1.1 Motivación y objetivos

El motivo de la realización de este proyecto es la implementación de un sistema capaz de comunicar cualquier FPGA con un ordenador, sin necesidad de utilizar un microcontrolador. Esta interfaz USB proporcionará una mayor velocidad de transferencia de datos entre la FPGA y el ordenador conectado.

Este sistema se ha desarrollado en hardware mediante su descripción en VHDL y su posterior prototipado en la FPGA utilizada. Además hace uso del dispositivo seleccionado FT120, capaz de conectar una interfaz USB a una entrada/salida de propósito general (GPIO) de una FPGA.

La elección del dispositivo utilizado se ha llevado a cabo en el estudio tecnológico [1] desarrollado con anterioridad, en el que se buscaba una alternativa sencilla a la utilización de un microcontrolador. El principal requisito fue la capacidad de transferir la información que se recibe desde un dispositivo USB a una interfaz paralela genérica. Para realizar esta elección se tuvieron en cuenta además otros aspectos como que la velocidad conseguida fuera superior a la de la interfaz serie, la simplicidad del dispositivo, el precio, la disponibilidad de drivers gratuitos, etc.

Una motivación para el desarrollo de este proyecto, es que algunas FPGAs no disponen de espacio suficiente para utilizar un microcontrolador que se encargue de comunicar una FPGA con un USB. Por ello se ha optado por implementar una alternativa más sencilla, permitiendo a todas las FPGAs utilizar una interfaz USB.

El principal objetivo consiste en que el sistema permita la comunicación a través de una interfaz USB a cualquier FPGA. Para ello, tal y como se ha indicado, se implementará en hardware el control de la comunicación con el dispositivo FT120, que funciona como controlador USB.

Como el objetivo de este proyecto requiere la implementación en hardware del protocolo USB, ha sido necesario un estudio detallado de su especificación [2]. Debido a su gran complejidad, no se ha fijado como objetivo su implementación completa, si no que se ha acotado al inicio de la comunicación. El resto del protocolo podrá ser completado en el futuro.

Tras analizar detalladamente los pasos necesarios para completar la inicialización de una conexión USB, se considera como objetivo suficiente para este proyecto la implementación del proceso de enumeración del dispositivo USB.

A parte del objetivo global de este proyecto, se han identificado una serie de objetivos intermedios que se deben cumplir indispensablemente para el desarrollo de este proyecto. Estos objetivos se enumeran y explican a continuación:

- Estudiar los pasos que el protocolo USB debe realizar para que se inicie la conexión entre el PC (*host*) y el dispositivo que se desea conectar, la FPGA en nuestro caso.

- Como se ha indicado anteriormente, para la implementación de este proyecto se ha utilizado el dispositivo FT120, por lo que ha sido necesario estudiar toda la información disponible al respecto. Lo fundamental ha sido aprender los comandos y datos con los cuales se gestiona el dispositivo y su correspondiente conexión USB.

- Implementar las acciones que se deben realizar para cumplir con el protocolo USB a través del dispositivo FT120:

    - Enviar un comando. Se debe conseguir que el FT120 reciba un comando enviado por el sistema implementado.

    - Enviar un dato. Se debe conseguir enviar un dato y que este sea leído por el dispositivo.

    - Recibir un dato. Se debe conseguir leer un dato enviado por el FT120 como respuesta a uno de los comandos recibidos.

    - Escribir *Buffer*. Se debe conseguir realizar una escritura del *buffer* para atender las solicitudes de información del dispositivo.

    - Leer *Buffer*. Se debe conseguir realizar una lectura del *buffer* para recibir los paquetes de *setup* enviados por el dispositivo.

1.2 Estructura de la memoria

En este apartado se explica cómo se encuentra organizado este documento y la información que se encontrará en cada uno de los capítulos en los que se divide.

1. **Introducción**. Se explican la motivación y los objetivos del proyecto, así como la estructura del mismo.
2. **Estado del Arte**. Se describen el protocolo USB, el dispositivo FT120 y las FPGAs, que son las bases fundamentales para la implementación y el desarrollo de este proyecto. El protocolo USB es lo que se quiere implementar, mientras el FT120 y la FPGA son los dispositivos utilizados para ello.
3. **Descripción del Diseño**. En este apartado se explica cómo se ha implementado el sistema, las interfaces de entrada/salida, los componentes que lo forman y las representaciones mediante diagramas de flujo y de estados (Principal, Ejecutar Comando, Leer *Buffer*, Escribir *Buffer*). En definitiva, se realiza una explicación del funcionamiento de cada una de las partes del sistema.
4. **Validación**. En primer lugar se muestran las simulaciones realizadas, que verifican la correcta implementación del sistema. Después se explican las pruebas a nivel hardware, con el sistema grabado en la FPGA, que comprueban su correcto funcionamiento. Por último, se detallan todas las dificultades encontradas durante la realización de este proyecto.
5. **Conclusiones**. Se explican los resultados obtenidos. Los objetivos conseguidos y los conocimientos adquiridos durante la realización de este proyecto.
6. **Trabajos Futuros**. Se explican las tareas que se deberán desarrollar en un futuro para completar la idea comenzada en este proyecto.
7. **Fases del proyecto**. Se detallan todas las fases que se han realizado hasta completar el sistema desarrollado. También se indica la duración de cada una de ellas.
8. **Presupuesto**. Se desglosan los gastos que suponen la realización de este proyecto.
9. **Glosario**. Se explican palabras que aparecen en el texto que pueden ser difíciles de comprender.
10. **Referencias**. Se indican las fuentes de información utilizadas.

2. Estado del arte

2.1 Protocolo USB

2.1.1 Introducción

USB es una interfaz serie que permite conectar un dispositivo a un ordenador y que se envíen datos entre ellos. USB ofrece una interfaz estándar para utilizar en muchos tipos diferentes de dispositivos.

Las principales características de USB son [1]:

- Banda de paso, disponibilidad desde algunos kilobits a varios megabits.
- Transferencia isócrona y asíncrona en el mismo bus.
- Varios tipos de periféricos en el mismo bus.
- Posibilidad de conectar hasta 127 periféricos.
- Tiempo de respuesta garantizado (para audio y vídeo).
- Flexibilidad a nivel de banda de paso.
- Fiabilidad, control de errores.
- Perfectamente integrado en el PC, *plug and play* (conectar y usar).
- Coste reducido en la versión de baja velocidad (1,5 Mbits/s).
- Posible expansión del bus.

La interfaz física [2] incluye la transmisión de energía eléctrica al dispositivo conectado. Las señales del USB son transmitidas por un cable de datos (par trenzado), con una impedancia característica de 90Ω, llamados D+ y D-. Utilizan señalización diferencial en *half-duplex* para combatir los efectos del ruido electromagnético en enlaces largos. D+ y D- usualmente operan en conjunto y no son conexiones simples. El reloj se transmite en el flujo de datos, la codificación es de tipo NRZI, existiendo un dispositivo que genera un bit de relleno (*bit stuffing*), que garantiza que la frecuencia de reloj permanezca constante. Las dos conexiones de los extremos son tierra y alimentación (5V). En la Figura 2.1.1 se puede observar la interfaz física.

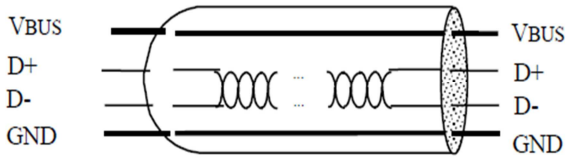


Figura 2.1.1 Cable USB [2]

2.1.2 Topología de bus

El USB [2] se presenta en forma de capas. Hay cuatro partes principales en la topología USB: *Host* y dispositivos (componentes principales), topología física (como se conectan los elementos), topología lógica (funciones y responsabilidades de los diferentes elementos) y relación entre el software del cliente y la función (como el software del cliente y las interfaces de las funciones relacionadas se ven mutuamente).

***Host* USB.** La composición lógica incluye un controlador, un software del sistema USB agregado y un cliente. Se trata de una entidad coordinadora con una posición física especial. Tiene responsabilidades específicas relacionadas con el USB y sus dispositivos conectados.

Dispositivo USB. Un dispositivo USB consta de una interfaz física USB, un dispositivo lógico USB y la función que realiza dicho dispositivo. El dispositivo USB proporciona funcionalidad adicional al *host*. Ayuda en la identificación y configuración de los dispositivos mediante información proporcionada (descriptores). Parte de la información que un dispositivo USB envía al *host* es común entre dispositivos lógicos, mientras que otra es específica de la funcionalidad proporcionada por el dispositivo.

Topología física. Los dispositivos se conectan al *host* mediante una topología en estrella. Los puntos de conexión son proporcionados por un tipo especial de dispositivo denominado *hub*, y se llaman puertos. Cada *hub* se sitúa en el centro de una estrella y cada segmento de cable es una conexión punto a punto. Los dispositivos que añaden funcionalidad adicional se denominan funciones. Estas conexiones pueden ser entre el *host* y un *hub* o función, o entre un *hub* y otro *hub* o función. Para evitar bucles se impone un ordenamiento por niveles obteniendo una configuración en forma de árbol. Debido a las limitaciones en los tiempos de propagación de los cables y los *hubs*, el número máximo de niveles son siete.

Topología lógica. El *host* se comunica con cada uno de los dispositivos lógicos como si dichos dispositivos estuvieran conectados directamente al puerto raíz. Para ello el *host* mantiene el conocimiento de la distribución física y así soportar la eliminación de los *hubs*, y los dispositivos conectados a través de él.

Relación entre el software del cliente y la función. Aunque se refleja la compartición del bus en la topología física y lógica, la manipulación de las interfaces de las funciones se presenta de distinta forma por parte del cliente software, ya que usan la interfaz de programación software USB para manipular sus funciones. Durante el funcionamiento, el software del cliente debe ser independiente de otros dispositivos que puedan estar conectados al USB.

2.1.3 Transferencias

USB es un bus compartido, por lo que un dispositivo normalmente no puede realizar una transferencia de un bloque completo (si podría si no hubiera más dispositivos). La transferencia estará dividida en tramas para compartir el ancho de banda con el resto de dispositivos del bus. Esto implica que la transferencia se completará en un periodo de tiempo superior que en el caso de que el bus no estuviera compartido.

Cada trama [3] estará compuesta de un Inicio de Trama (SOF), seguido de una o más transacciones. Las tramas serán enviadas en intervalos de 1 ms para *full-speed* (125 µs para *high-speed*). Cada transacción a su vez estará compuesta de una serie de paquetes. Estos comienzan con un campo de sincronización SYNC y terminan con un código de fin de paquete EOP. Las transacciones están compuestas de un paquete *token* (esto como mínimo), de uno o

más paquetes de datos opcionales y un paquete de *handshake*. En la Figura 2.1.2 se puede observar la evolución temporal de una comunicación USB.

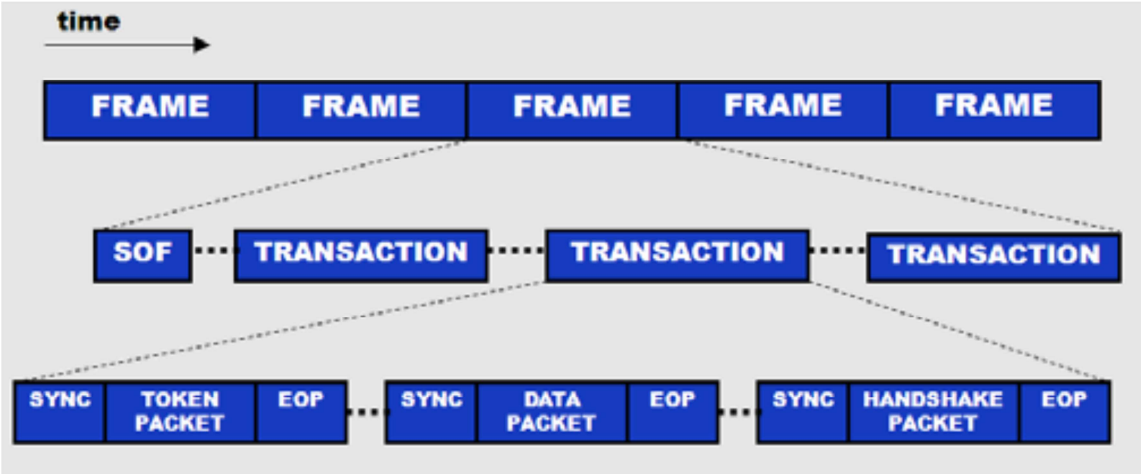


Figura 2.1.2 Comunicación USB [3]

El *host* es el encargado de iniciar todas las transferencias. Los dispositivos permanecen a la espera hasta recibir un paquete toquen dirigido a él, en el cuál se indica el tipo de transferencia a realizar.

2.1.3.1 Elementos de una transferencia

Cada transferencia USB [4] consiste en una o más transacciones, y cada transacción a su vez contiene paquetes que contienen información. Para comprender las transacciones, los paquetes y su contenido, primero es necesario conocer los *endpoints* y las *pipes.*

Endpoints

Un *endpoint* [2] es una porción única direccionable de un dispositivo USB que es origen o destino de información en un flujo de comunicación entre el *host* y el dispositivo. También se puede definir el *endpoint* [4] como un *buffer* que guarda múltiples bytes, es decir, es un bloque de memoria de datos o un registro en el chip del controlador. Los datos almacenados en un *endpoint* son datos recibidos o datos esperando a ser enviados. El *host* también tiene *buffers* que mantienen los datos recibidos y los que van a ser enviados, pero no tiene *endpoints*.

Cada dispositivo [2] tiene una dirección única asignada por el sistema durante el periodo de conexión del dispositivo. Cada *endpoint* del dispositivo tendrá un identificador único llamado número de *endpoint*, el cual se asignará durante el periodo de diseño. Cada *endpoint* tiene además una dirección del flujo de datos determinada. La combinación de la dirección del dispositivo, número de *endpoint* y dirección permite que cada *endpoint* esté referenciado de forma exclusiva. En resumen, cada *endpoint* es una conexión simple que soporta el flujo de datos en una dirección: o bien de entrada (desde el dispositivo al *host*) o de salida (desde el *host* al dispositivo).

Como el bus es controlado por el *host*, no es posible escribir directamente en él, lo que se hace es escribir datos en un *endpoint* de entrada (IN), donde los datos son almacenados en el *buffer* hasta que el *host* envía un paquete de tipo IN a ese *endpoint* solicitando los datos.

Todos los dispositivos están obligados a tener un método de control por defecto que utiliza un *endpoint* de entrada y otro de salida con número *endpoint* 0. Esto es utilizado para llevar a cabo la configuración del dispositivo. Estos *endpoint*s son siempre accesibles una vez se acopla el dispositivo, se alimenta y recibe un reseteo del bus.

Las funciones de los dispositivos pueden tener los *endpoint*s adicionales necesarios para su implementación. Los dispositivos de baja velocidad, están limitados a dos *endpoints* opcionales a parte de los dos requeridos para implementar la *pipe* de control por defecto. Los dispositivos *full speed* pueden tener tantos como permite la definición del protocolo (15 *endpoints* de entrada y 15 de salida). Estos no se pueden usar hasta que el dispositivo es configurado como una parte normal del proceso de configuración del dispositivo.

Pipes

Una *pipe* [2] es un enlace virtual entre un *endpoint* de un dispositivo y el software del *host*, que permite mover los datos de uno a otro a través del *buffer* de memoria. Este enlace tiene unos parámetros asociados como el ancho de banda, el tamaño de los paquetes, el tipo de transferencia y la dirección de flujo.

El *host* [4] establece las *pipes* durante la enumeración. Si el dispositivo es removido del bus, el *host* remueve las *pipes* que ya no son necesarias. El *host* puede pedir nuevas *pipes* o remover las que ya no se utilizan en otras ocasiones mediante la solicitud de una nueva configuración alternativa o interface para un dispositivo. Cada dispositivo tiene una *pipe* de control por defecto que usa el *endpoint* 0.

Hay dos tipos [2] posibles de *pipes*: *Stream*, que no tiene un formato USB definido y se usa en transferencias de tipo bulk, isócronas e interrupción; y pipes de tipo *Message*, que si tiene un formato USB definido y se usa en transferencias de control.

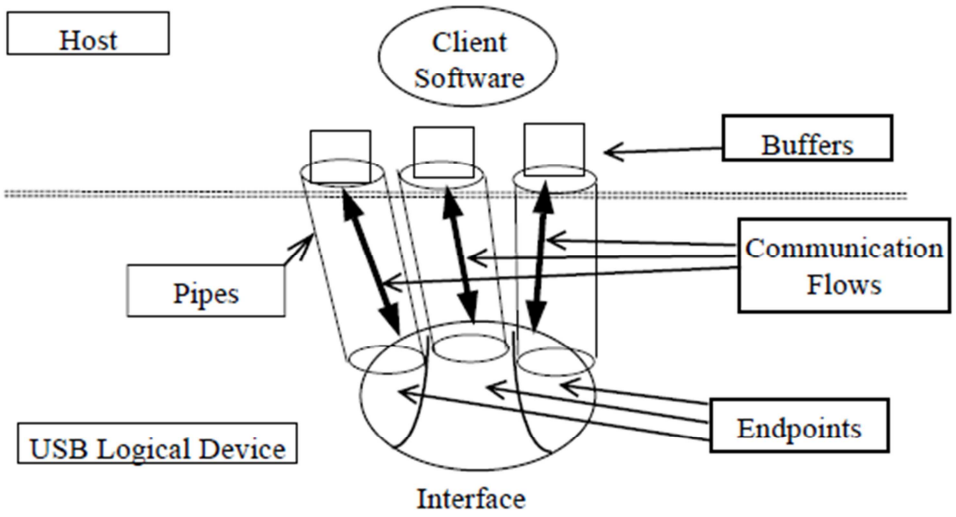


Figura 2.1.3 Flujo de comunicación USB [2]

La Figura 2.1.3 muestra como los flujos de comunicación se realizan a través de *pipes* entre *endpoints* y los *buffers* de memoria del *host*. El software del *host* se comunica con el dispositivo lógico a través de un conjunto de flujos de comunicación. Estos son elaborados por

el diseñador de software/hardware del dispositivo para que las características de transferencia USB se correspondan con las necesidades de comunicación del dispositivo.

2.1.3.2 Tipos de transferencias

El protocolo USB [2] debe poder utilizarse por distintos tipos de dispositivos, los cuales necesitan diferentes características del flujo de comunicación: formato de datos, dirección del flujo, tamaño de paquete, limitaciones de acceso al bus, limitaciones de latencia, secuencia de datos requeridos y corrección de errores. Por este motivo, existen cuatro tipos de transferencias para adecuarse a las necesidades de los distintos dispositivos.

Los cuatro tipos de transferencias y sus principales características son los siguientes:

Transferencias de Control. Las comunicaciones de solicitud/respuesta se realizan en ráfagas, no periódicas e iniciadas por el software del *host*. Estas transferencias se utilizan para configurar el dispositivo, obtener información sobre el dispositivo, enviar comandos al dispositivo o recuperar informes sobre la situación del dispositivo. Todos los dispositivos USB deben soportar este tipo de transferencias y deben estar garantizadas mediante la disponibilidad de suficiente ancho de banda.

Transferencias Isócronas. Este tipo de comunicación entre el *host* y el dispositivo es periódica y continua. Se suele utilizar para la información relevante en el tiempo (en tiempo real), ya que tienen un tiempo de envío garantizado. Por otro lado, no presenta control de errores y sólo pueden utilizarlo los dispositivos *full* y *high-speed*.

Transferencias de Interrupción. Estas comunicaciones tienen latencia delimitada y baja frecuencia, pero garantizan la transferencia de pequeñas cantidades de datos. Este tipo de transferencias son la única forma de que los dispositivos *low-speed* envíen datos.

Transferencias Bulk. Las comunicaciones de este tipo son no periódicas y con grandes cantidades de datos enviados a ráfagas. Normalmente se utiliza para datos que pueden usar cualquier ancho de banda disponible y también pueden retrasarse hasta que el ancho de banda esté disponible. La entrega de los datos está garantizada, pero no hay una garantía de latencia máxima ni ancho de banda. Sólo los dispositivos *full* y *high-speed* pueden utilizarlas.

2.1.4 Enumeración

2.1.4.1 Estados del dispositivo USB

Se definen [2] seis estados en los que puede encontrarse un dispositivo: *Attached*, *Powered*, *Default*, *Address*, *Configured* y *Suspended*. Para que una función de un dispositivo pueda utilizarse debe cumplirse que esté conectado al puerto USB, esté alimentado, haya sido reseteado, tenga una dirección asignada, esté configurado y no esté suspendido.

El siguiente diagrama muestra los estados de un dispositivo:

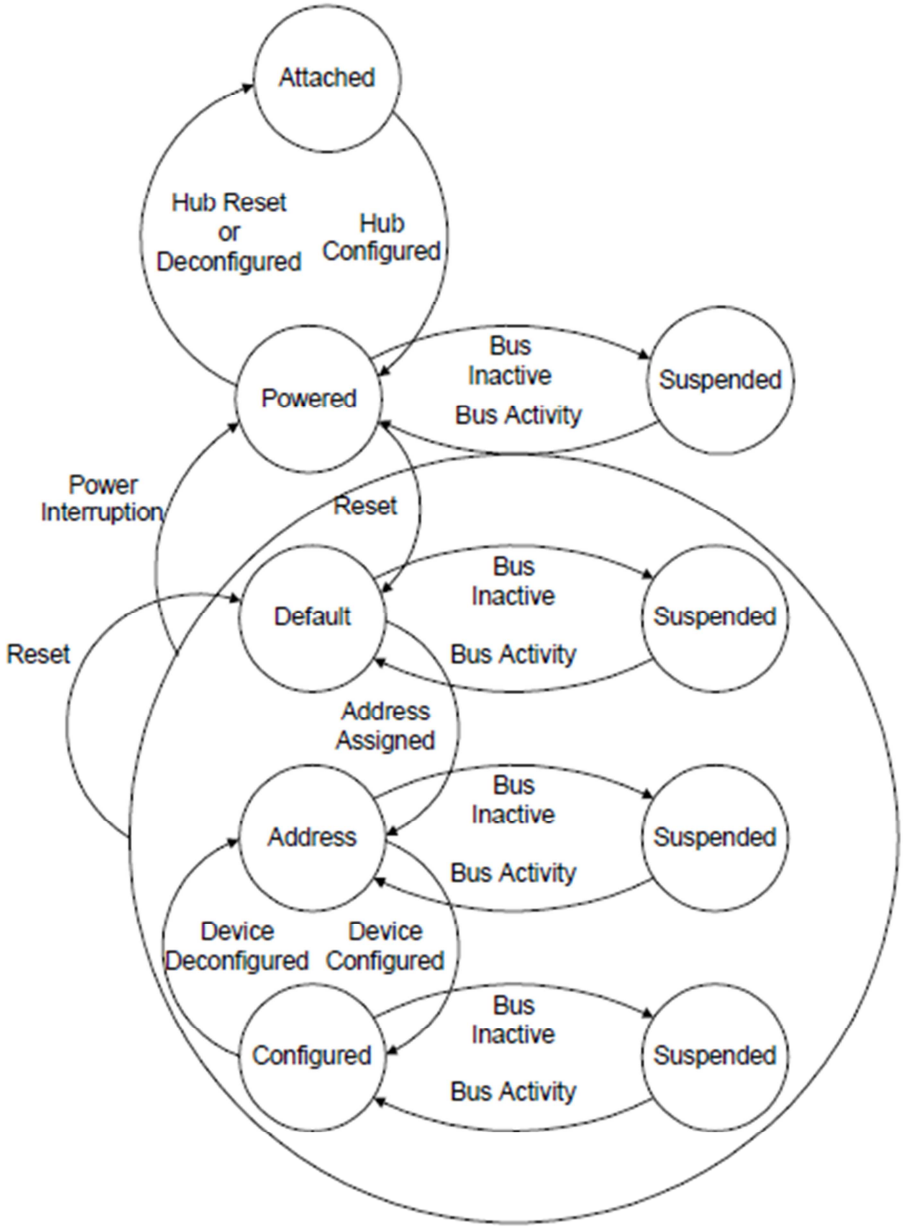


Figura 2.1.4 Diagrama de estados del dispositivo [2]

Una vez conectado y alimentado el dispositivo, éste no responde a las transacciones hasta que ha sido reseteado. Después de recibir el *reset*, el dispositivo utilizará la dirección predeterminada. La velocidad que utilizará el dispositivo, *low* o *full-speed*, viene determinada por las resistencias de terminación.

Los dispositivos usan la dirección por defecto hasta que el *host* le asigna una dirección única. Esta dirección se mantiene incluso cuando el dispositivo está suspendido, además continúa respondiendo a la dirección por defecto.

Para que la función del dispositivo pueda ser utilizada es necesario configurarlo. Desde la perspectiva del dispositivo, la configuración implica procesar correctamente una petición *Set Configuration* con un valor distinto de cero.

Con el fin de ahorrar energía, los dispositivos entran en el estado suspendido cuando pasa cierto tiempo sin tráfico en los buses. El dispositivo mantiene el estado interno, incluyendo dirección y configuración.

2.1.4.2 Proceso de enumeración

Las comunicaciones USB [4] se pueden clasificar en dos categorías atendiendo a la utilidad que se hace de ellas. Por un lado están las comunicaciones usadas para enumerar y configurar los dispositivos USB, de esta manera el *host* aprende lo necesario del dispositivo y lo prepara para su utilización. Por otro están las comunicaciones que llevan a cabo los propósitos de los dispositivos, es decir, se lleva a cabo un intercambio de datos que realiza la función para la cual el dispositivo fue diseñado. Teniendo en cuenta que la primera categoría es común a todos los dispositivos USB se procederá a explicar con mayor detalle.

Los pasos siguientes son una secuencia típica de sucesos que se producen durante el proceso de enumeración. El *firmware* del dispositivo no debe asumir que las solicitudes y eventos de la enumeración ocurrirán en un orden determinado. Un dispositivo deberá detectar y responder a cualquier solicitud de control u otro evento del bus en cualquier momento.

1. **El usuario conecta un dispositivo a un puerto USB.** El *hub* proporciona energía al puerto, y el dispositivo está en el estado *Powered.*

2. **El *hub* detecta el dispositivo.** La detección de la conexión se realizará cuando se produzca una transición de voltaje en las líneas D+ y D-.

3. **El *host* aprende un nuevo dispositivo.** Cada *hub* usa un *endpoint* de interrupción para informar de los eventos en el *hub*. El informe indica únicamente si el *hub* o el puerto ha experimentado algún evento. En el aprendizaje de un evento, el *host* envía al *hub* una solicitud *Get Port Status* para averiguar más. La información devuelta indica al *host* cuando un dispositivo está recién conectado.

4. **El *hub* detecta si un dispositivo es *low* o *full speed*.** Antes de que el *hub* resetee el dispositivo, el *hub* determina si el dispositivo es *low* o *full speed* examinando los voltajes de las dos líneas. El *hub* detecta la velocidad de un dispositivo determinando que línea tiene el voltaje más alto cuando está inactivo. El *hub* envía la información al *host* en respuesta a la solicitud *Get Port Status.* USB 2.0 requiere que la detección de la velocidad se produzca antes del *reset*, por lo que el *hub* sabe si hay que comprobar si el dispositivo es de alta velocidad durante el reseteo.

5. **El *hub* resetea el *dispositivo.*** Cuando un *host* aprende un nuevo dispositivo, el controlador del *host* envía al *hub* una solicitud *Set Port Feature* que pide al *hub* que resetee el puerto. El *hub* pone las líneas de datos USB del dispositivo en la condición de reinicio durante al menos 10 milisegundos. *Reset* es una condición especial en el que tanto D+ y D- están a bajo nivel lógico (normalmente, las líneas tienen niveles lógicos opuestos). El *hub* envía el *reset* sólo a los dispositivos nuevos.

6. **El *host* aprende si un dispositivo *full-speed* soporta *high-speed*.** Para detectar si un dispositivo soporta *high-speed* se usan dos estados de señal especiales. En el estado Chirp J, solo la línea D+ es activada y en el estado Chirp K, solo la línea D- es activada.

El dispositivo que soporta *high speed* envía un Chirp K, el *hub* lo detecta y responde con una secuencia KJKJKJ.

7. **El *hub* establece el recorrido de la señal entre el dispositivo y el bus.** El *host* verifica que el dispositivo ha salido del estado de *reset* mediante el envío de una solicitud *Get Port Status.* Un bit en los datos devueltos indica si la enumeración continúa en estado de *reset* o no. Si es necesaria dicha solicitud se repite hasta que se haya salido de dicho estado. Entonces el dispositivo está listo para responder a las transferencias de control al *endpoint* 0. El dispositivo se comunica con el *host* usando la dirección por defecto 00h.

8. **El *host* envía una solicitud *Get Descriptor* para aprender el tamaño máximo de paquete de la *pipe* por defecto.** El *host* envía la solicitud a la dirección 0 del dispositivo, *endpoint* 0. Debido a que el *host* enumera solo un dispositivo a la vez, solo un dispositivo responderá a las comunicaciones dirigidas a la dirección del dispositivo 0. El descriptor es una estructura de datos que almacena información sobre los dispositivos USB. El descriptor contiene el tamaño máximo de paquete en el octavo byte, por lo que un dispositivo está obligado a responder con al menos 8 byte para que figure dicha información. En este punto, el *host* puede pedir al *hub* que reinicie el dispositivo, como en el paso 5, aunque según la especificación no se requiere.

9. **El *host* asigna una dirección.** El controlador del *host* asigna una dirección única al dispositivo mediante el envío de la solicitud *Set Address*. En este momento las solicitudes dejarán de ser a través de la dirección por defecto, y se implementará la nueva dirección. El dispositivo está en el estado *Address.* La dirección es válida hasta que se desconecta el dispositivo, se resetea el puerto o se reinicia el sistema.

10. **El *host* aprende las capacidades del dispositivo.** El *host* envía una solicitud *Get Descriptor* a la nueva dirección para leer el descriptor del dispositivo. Esta vez el *host* procede a leer el descriptor en su totalidad. El descriptor contiene el tamaño máximo de paquete, el número de configuraciones que soporta el dispositivo, y otra información básica sobre el dispositivo. El *host* utiliza esta información en las siguientes comunicaciones. El *host* seguirá aprendiendo sobre el dispositivo mediante la solicitud de uno o más descriptores de configuración, especificados en el descriptor de dispositivo. En esta segunda petición también suelen ser enviados los descriptores de interfaz y de *endpoint*.

11. **El *host* asigna y carga un controlador de dispositivo.** El *host* busca la mejor elección de controlador de dispositivo para gestionar las comunicaciones con el dispositivo. Después de que el sistema operativo asigne y cargue el controlador, el controlador puede solicitar al dispositivo que reenvíe los descriptores o envíe otros descriptores específicos de clase. Como excepción, el *host* sólo puede asignar los drivers tras la configuración del dispositivo para el caso de dispositivos compuestos.

12. **El controlador de dispositivo del *host* selecciona una configuración.** Después de aprender sobre el dispositivo a partir de los descriptores, el controlador de dispositivo solicita una configuración. Para ello el *host* envía una solicitud *Set Configuration* con el número de configuración deseado. El dispositivo lee la petición y habilita la

configuración solicitada. El dispositivo está ahora en el estado *Configured* y las interfaces del dispositivo están habilitadas.

2.1.4.3 Descriptores

Los dispositivos USB [2] reportan sus atributos utilizando descriptores. Un descriptor es una estructura de datos con un formato definido. Cada descriptor empieza con un campo que contiene el número total de bytes en el descriptor seguido por un campo que identifica el tipo de descriptor. Durante la enumeración, el *host* usa transferencias de control para solicitar descriptores a un dispositivo.

Hay cinco tipos de descriptores que irán proporcionando la información necesaria a medida que avanza la enumeración (dispositivo, configuración, interfaz, *endpoint* y cadena).

Descriptor de Dispositivo. El descriptor de dispositivo es el primero en leerse y describe la información general acerca del dispositivo. Incluye información que se aplica a nivel global para el dispositivo y todas sus configuraciones. Un dispositivo USB tiene un único descriptor de dispositivo. La información que proporciona es sobre el fabricante, número de producto, número de serie, clase de dispositivo y número de configuración.

Descriptor de Configuración. El descriptor de configuración describe información acerca de una configuración de dispositivo específico. La información que proporciona es sobre la alimentación del dispositivo y sobre las interfaces que son soportadas.

Descriptor de Interfaz. El descriptor de interfaz describe una interfaz específica dentro de una configuración, ya que puede haber varias. Además también informa del número de *endpoints* usados en la interfaz.

Descriptor de *Endpoint*. El descriptor de *endpoint* identifica el tipo de transferencia y su sentido. Contiene la información requerida por el *host* para determinar los requisitos de ancho de banda de cada *endpoint*.

Descriptor de Cadena (*string*). El descriptor de *string* es opcional, pero los campos de referencia en ellos son obligatorios. Si un dispositivo no admite descriptores de *string* los campos estarán puestos a cero.

2.2 FT120

2.2.1 Introducción

El FT120 [5] proporciona un controlador de dispositivo *full-speed* USB optimizado en coste y características. Este permite la comunicación con un microcontrolador o FPGA a través de una interfaz genérica paralela, ofreciendo conectividad USB con los *endpoints bulk*, isócronos y de interrupción. Tiene las siguientes características avanzadas:

- Compatible con USB 2.0 *Full Speed.*
- Controlador de dispositivo USB de alto rendimiento con SIE (motor de interfaz serie) integrada, *buffer endpoint*, transceptor y regulador de voltaje.
- Soporta interfaz paralela de 8 bits para un microcontrolador externo.
- Soporta operación de DMA.
- Soporte para múltiples modos de interrupción para facilitar las transferencias masivas (*bulk*) e isócronas.
- Motor DMA completamente autónomo para la transferencia rápida de datos.
- Soporta interfaz en paralelo de alta velocidad para un microcontrolador externo.
- Bus de direcciones/datos separados o bus de datos/direcciones multiplexado.
- Integrados 320 bytes de *buffer* de *endpoint* configurables.
- Esquema de *buffer* doble para el *endpoint* principal mejorando la velocidad de transferencia de datos.
- Pin de salida de reloj con frecuencia programable (4 - 24 MHz).
- Reloj de salida de 30 kHz durante la suspensión.
- Integrada resistencia de *pull-up* D+ para la conexión USB.
- Indicador de conexión USB que parpadea con las transmisiones y recepciones USB.
- Soporta aplicaciones alimentadas por bus o autoalimentadas.
- Funciona con una sola fuente de alimentación a 3.3V o 5V.
- Reguladores internos LDO (baja caída de tensión) de 1.8V y 3.3V.
- Circuito integrado de encendido-reinicio.
- Compatible con controladores de *host* UHCI/OHCI/EHCI.
- Rango de temperatura de funcionamiento de -40°C a 85°C.
- Disponible en encapsulado Pb-free TSSOP-28 y QFN-28.

Las aplicaciones típicas son:

- Proporcionar conexión USB a microcontroladores.
- Control industrial USB.
- Transferencia de datos para almacenamiento masivo en multitud de aplicaciones de sistemas embebidos, médicas, registro de datos industriales, medidas de potencia e instrumentación de pruebas.
- Proporciona conexión USB a FPGAs.
- La utilización USB para añadir modularidad al sistema.
- Soporta transferencias isócronas para aplicaciones de video, control industrial e inspección de calidad.

2.2.2 Diagrama de bloques

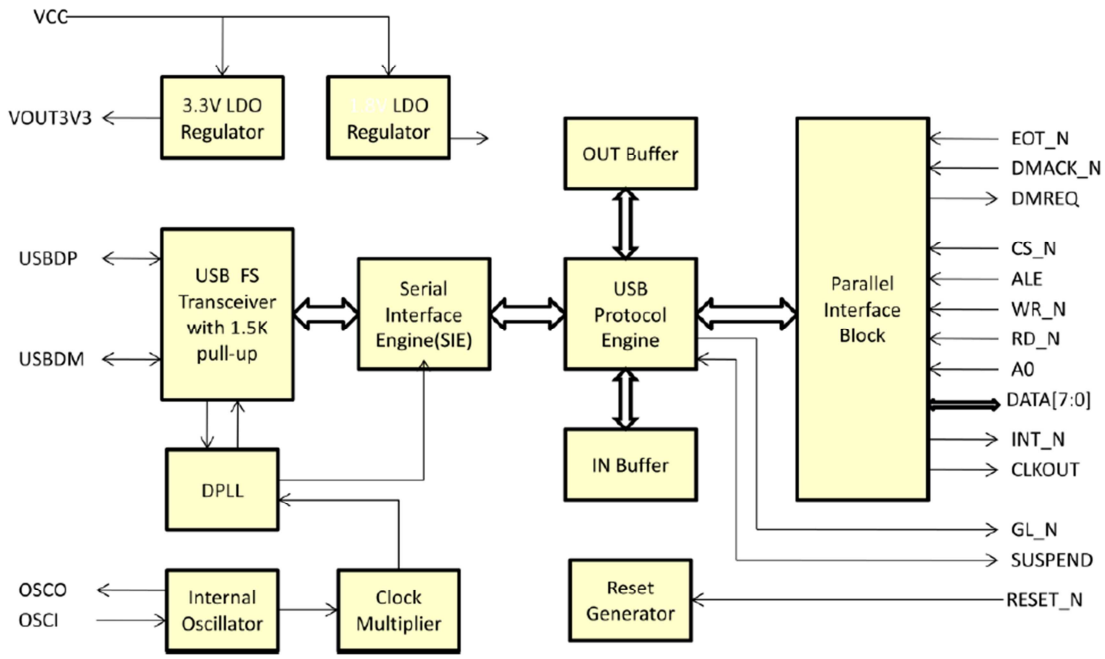


Figura 2.2.1 Diagrama de bloques del FT120 [5]

Regulador LDO 1.8V. El regulador LDO (Low-Dropout) de 1.8V genera el voltaje de referencia para el núcleo interno de los circuitos integrados con capacidades de entrada de 3.3V o 5V.

Regulador LDO 3.3V. El regulador LDO de 3.3V genera la fuente de voltaje de 3.3V para el transceptor. Un condensador externo de desacoplo es necesario conectarlo al pin de salida del regulador VOUT3V3. El regulador solo proporciona 3.3V a la resistencia interna de *pull-up* de 1.5kΩ en el pin USBDP. Las tensiones de alimentación permitidas son 5V o 3.3V. Cuando se usa 3.3V como entrada de voltaje, los pines VCC y VOUT3V3 deben ser conectados. Esto equivaldrá a omitir el regulador.

Transceptor USB. El transceptor USB proporciona la interfaz física *full-speed* para USB 1.1/USB 2.0. Los drivers de salida proporcionan control de nivel de *slew rate* de 3.3V, mientras una entrada diferencial y dos receptores de entrada de un solo extremo proporcionan datos de entrada. Una resistencia de *pull-up* de 1.5kΩ está incorporada en USBDP.

DPLL. La celda DPLL bloquea a la entrada los datos USB NRZI y genera el reloj recuperado y las señales de datos.

Oscilador interno. El oscilador interno genera un reloj de referencia de 6MHz a partir del cristal de 6MHz. El oscilador solo tiene la capacidad de funcionar a partir de un reloj externo aplicado en el pin OSCI. Este proporciona una entrada a la función de multiplicador de reloj.

Multiplicador de reloj. Las señales de reloj de referencia de 12MHz y 48MHz para diversos bloques internos pueden ser generadas a partir de los 6MHz proporcionados por el oscilador y el circuito multiplicador de reloj.

Motor de interfaz serie (SIE). La SIE se encarga de convertir los datos USB de paralelo a serie y de serie a paralelo. De acuerdo con la especificación USB 2.0, se lleva a cabo la generación del relleno de bits y del CRC5/CRC16. También chequea el CRC de los flujos de datos USB.

Motor del protocolo USB. El motor del protocolo USB gestiona el flujo de datos desde el *endpoint* de control del dispositivo USB. Se ocupa de las solicitudes del protocolo USB a bajo nivel generadas por el controlador de *host* USB. El motor del protocolo también incluye una unidad de gestión de memoria que se ocupa de los *buffer* de *endpoint*.

***Buffer* de salida.** Los datos enviados desde el controlador del *host* USB al FT120 a través del *endpoint* de salida de datos USB se almacenan en el *buffer* de salida. Los datos se eliminan del *buffer* de salida y pasan a la memoria del sistema bajo el control del bloque de interfaz paralela.

***Buffer* de entrada.** Los datos de la memoria del sistema se guardan en el *buffer* de entrada. El controlador del *host* USB elimina los datos del *buffer* de entrada enviando una solicitud USB de datos desde el *endpoint* de entrada de datos del dispositivo.

Generador de reinicio. El generador de reinicio integrado proporciona un reinicio fiable de los circuitos internos del dispositivo al encenderlo. El pin de entrada RESET_n permite a un dispositivo externo reiniciar el FT120.

Bloque de interfaz paralela. El bus paralelo de 8 bits permite la interfaz directa a un microcontrolador genérico (MCU), soportando la configuración del bus tanto multiplexando direcciones/datos, como no multiplexándolos. El FT120 también soporta operaciones de acceso directo a memoria (DMA). Con este acceso de datos DMA se puede escribir en el *buffer* de entrada o leer del *buffer* de salida sin la intervención del MCU. El acceso DMA puede hacerse en un solo ciclo o en modo ráfaga.

2.2.3 Modos de Interrupción

EL pin de interrupción [5] del FT120 (INT_n) puede ser programado para generar interrupciones en diferentes modos. La fuente de interrupción puede ser cualquier bit del registro de interrupción, o recibiendo un paquete SOF, o ambos. Los modos de interrupción son seleccionados por dos bits del registro, uno es el bit SOF-only del modo de interrupción (bit 7 del registro de factor de división del reloj), y el otro es el bit del modo del pin de interrupción (bit 5 del registro de configuración DMA).

Interrupt mode	Bit SOF-only Interrupt Mode	Bit Interrupt Pin Mode	Interrupt source
0	0	0	Any bit in Interrupt register
1	0	1	Any bit in Interrupt register and SOF
2	1	X	SOF only

Tabla 2.2.1 Modos de interrupción [5]

2.2.4 Gestión del *Buffer* de *endpoint*

El FT120 [5] tiene 3 *endpoints* físicos (EP0, EP1 y EP2) o 6 *endpoints* lógicos (EPI0-EPI5). EP0 es un *endpoint* de control, con tamaño de paquete máximo 16 bytes tanto para el *endpoint* de salida de control (EPI0) como el *endpoint* de entrada de control (EPI1). EP1 puede ser usado como *endpoint bulk* o como *endpoint* de interrupción, con un tamaño máximo de paquete de 16 bytes, para ambos *endpoints* de salida (EPI2) y de entrada (EPI3). La siguiente tabla muestra el tipo de *endpoint* y los correspondientes tamaños máximos de paquete.

Endpoint Number	Endpoint Index (EPI)	Endpoint Direction	Transfer Type	Max Packet Size
0	0	OUT	Control	16
	1	IN	Control	16
1	2	OUT	Bulk/Interrupt	16
	3	IN	Bulk/Interrupt	16

Tabla 2.2.2 Configuración del *endpoint* EP0 y EP1 [5]

EP2 es el *endpoint* primario. Este puede ser configurado como un *endpoint bulk*/interrupción o isócrono. El tamaño máximo de paquete permitido para el EP2 depende del modo de configuración indicado en el comando *Set Mode*. La siguiente tabla muestra todos los modos de configuración del EP2.

Endpoint Configuration Mode (EP2)	Endpoint Index (EPI)	Endpoint Direction	Transfer Type	Max Packet Size
0	4	OUT	Bulk/Interrupt	64
(default)	5	IN	Bulk/Interrupt	64
1	4	OUT	Isochronous	128
2	5	IN	Isochronous	128
3	4	OUT	Isochronous	64
	5	IN	Isochronous	64

Tabla 2.2.3 Configuración del *endpoint* EP2 [5]

Como *endpoint* primario, EP2 es adecuado para la transmisión y recepción de datos relativamente largos. Para mejorar el rendimiento de los datos, un par de *ping-pong buffer* se implementan para el *buffer* de EP2. Esto permite la operación concurrente entre el acceso al bus UBS y el acceso al bus local MCU o DMA. Por ejemplo, para el *endpoint* EP2 de entrada (EPI5), el *host* USB puede leer datos desde el *buffer ping* del FT120, mientras el MCU local está escribiendo al mismo tiempo en el *buffer pong*. El *host* USB puede leer más tarde desde el *pong buffer* del FT120 sin esperar a que este se llene. La conmutación del *buffer* es manejada automáticamente por el FT120.

El *buffer* EP2 solo soporta operaciones DMA. El MCU necesita inicializar la operación DMA a través del comando *Set DMA*. Una vez que la operación DMA está habilitada, los datos se moverán entre la memoria del sistema y el *buffer* del *endpoint* del FT120 según el controlador DMA. La conmutación del *buffer* entre el *ping buffer* y el *pong buffer* es manejada automáticamente.

2.2.5 Comandos

Los comandos [5] soportados por el FT120 se encuentran resumidos en la siguiente tabla. Se incluyen comandos de inicialización, comandos de flujos de datos y comandos genéricos.

Command Name	Target	Code (hex)	Data phase
Initialization Commands			
Set Address Enable	Device	D0h	Write 1 byte
Set Endpoint Enable	Device	D8h	Write 1 byte
Set Mode	Device	F3h	Write 2 bytes
Set DMA	Device	FBh	Write/Read 1 byte
Data Flow Commands			
Read Interrupt Register	Device	F4h	Read 2 bytes
Select Endpoint	Endpoint 0 OUT	00h	Read 1 byte (optional)
	Endpoint 0 IN	01h	Read 1 byte (optional)
	Endpoint 1 OUT	02h	Read 1 byte (optional)
	Endpoint 1 IN	03h	Read 1 byte (optional)
	Endpoint 2 OUT	04h	Read 1 byte (optional)
	Endpoint 2 IN	05h	Read 1 byte (optional)
Read Last Transaction Status	Endpoint 0 OUT	40h	Read 1 byte
	Endpoint 0 IN	41h	Read 1 byte
	Endpoint 1 OUT	42h	Read 1 byte
	Endpoint 1 IN	43h	Read 1 byte
	Endpoint 2 OUT	44h	Read 1 byte
	Endpoint 2 IN	45h	Read 1 byte

Read Endpoint Status	Endpoint 0 OUT	80h	Read 1 byte
	Endpoint 0 IN	81h	Read 1 byte
	Endpoint 1 OUT	82h	Read 1 byte
	Endpoint 1 IN	83h	Read 1 byte
	Endpoint 2 OUT	84h	Read 1 byte
	Endpoint 2 IN	85h	Read 1 byte
Read Buffer	Selected Endpoint	F0h	Read multiple bytes
Write Buffer	Selected Endpoint	F0h	Write multiple bytes
Set Endpoint Status	Endpoint 0 OUT	40h	Write 1 byte
	Endpoint 0 IN	41h	Write 1 byte
	Endpoint 1 OUT	42h	Write 1 byte
	Endpoint 1 IN	43h	Write 1 byte
	Endpoint 2 OUT	44h	Write 1 byte
	Endpoint 2 IN	45h	Write 1 byte
Acknowledge Setup	Selected Endpoint	F1h	None
Clear Buffer	Selected Endpoint	F2h	None
Validate Buffer	Selected Endpoint	FAh	None
General Commands			
Read Current Frame Number	Device	F5h	Read 1 or 2 bytes
Send Resume	Device	F6h	None

Tabla 2.2.4 Conjunto de comandos FT120 [5]

2.2.6 Temporización de Escritura y Lectura

La información [5] de temporización del dispositivo FT120, se muestra en la figura 2.2.2. En ella se puede observar cómo se deben realizar las escrituras y lecturas del dispositivo, indicando cuando y que valor deben tener las señales de entrada que correspondan.

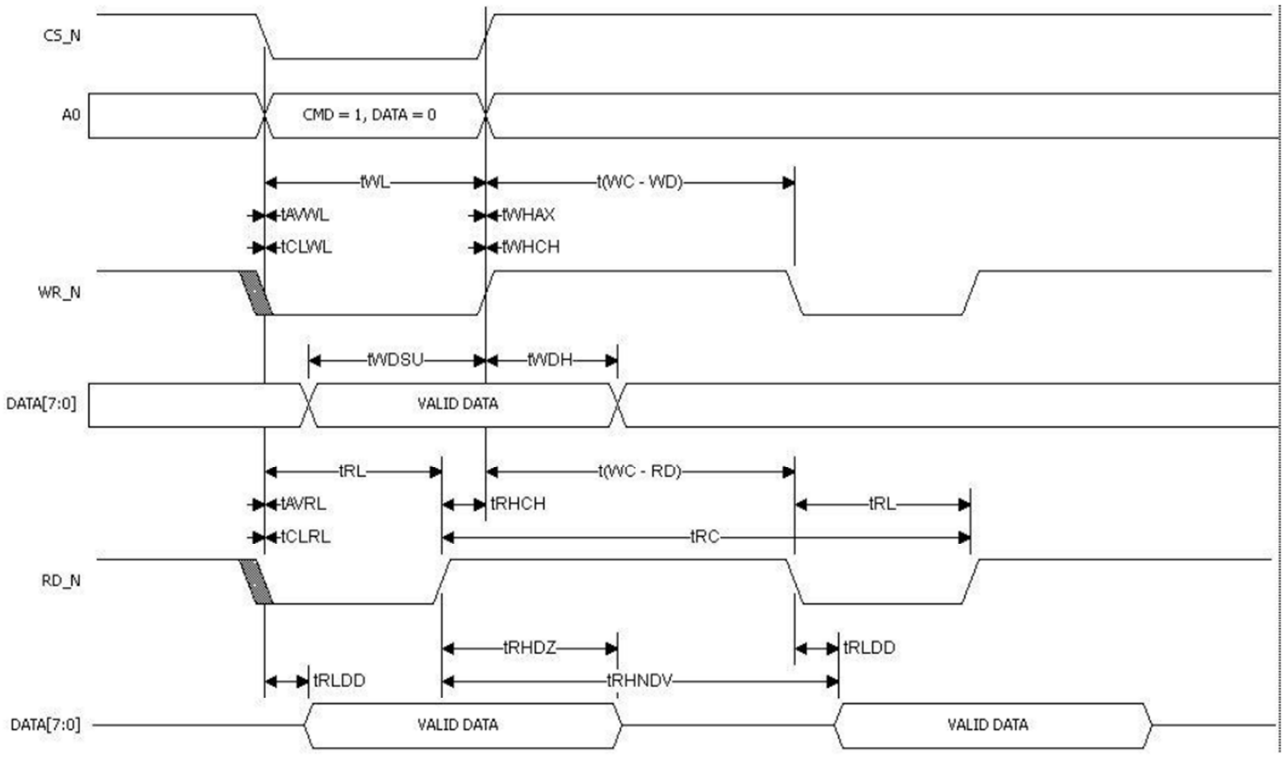


Figura 2.2.2 Temporización de interfaz paralela [5]

En la siguiente tabla 2.2.5 se muestran los valores que deben tener los tiempos de la anterior figura.

Escribir	Mínimo	Máximo
tWL	20ns	-
tWDSU	30ns	-
tWDH	10ns	-
t(WC-WD)	600ns	-
Leer	Mínimo	Máximo
tRHCH	5ns	-
tRL	20ns	-
tRLDD	-	20ns
tRHDZ	-	20ns
tRC	500ns	-
t(WC-RD)	600ns	-

Tabla 2.2.5 Temporización de interfaz paralela [5]

2.3 Kit de evaluación de la serie FT12

El kit de evaluación de la serie FT12 [6] se usa para desarrollar y demostrar las funciones de los circuitos integrados FTDI FT120, FT121 y FT122. El kit consiste en una placa principal y uno de los módulos adicionales FT120/FT121/FT122. Estos módulos pueden proporcionar una función de dispositivo USB genérico a un microcontrolador (MCU) con las siguientes interfaces:

- UMFT120DC: interfaz 8051 de 8bits, 3 *endpoints* bidireccionales.
- UMFT121DC: interfaz esclava SPI, 8 *endpoints* bidireccionales.
- UMFT122DC: interfaz 8051 de 8bits, 8 *endpoints* bidireccionales.

Características:

- Conector micro-b USB para conectarse al *host* USB o como puerto de carga.
- Permite configurar alimentación por bus o alimentación propia.
- Microcontrolador LPC1114 Cortex-M0 para un desarrollo fácil del software.
- Pulsador y leds para aplicaciones HID.
- Sistema con firmware actualizable.
- Área en el prototipo de la placa para el desarrollo de aplicaciones.

2.3.1 Descripción Hardware

2.3.1.1 Placa principal UMFT12XEV

La placa UMFT12XEV [6] está destinada a ser utilizada como una plataforma hardware para permitir una fácil evaluación de las series FTDI FT120, FT121 y FT122 de controladores periféricos USB. Las placas UMFT12XEV utilizan un microcontrolador (LPC1114) basado en NXP Cotex-M0 para interconectarse a la serie FT12 con el bus paralelo o SPI. Los Leds y pulsadores pueden ser usados para demostrar las funciones del teclado HID (dispositivos de interfaz humana). Un área del prototipo es también construido sobre la placa UMFT12XEV, permitiendo que se añada un circuito de aplicación definida por el usuario de manera que la aplicación *hardware/firmware* pueda ser desarrollada y probada en el prototipo antes de pasar a la producción en masa.

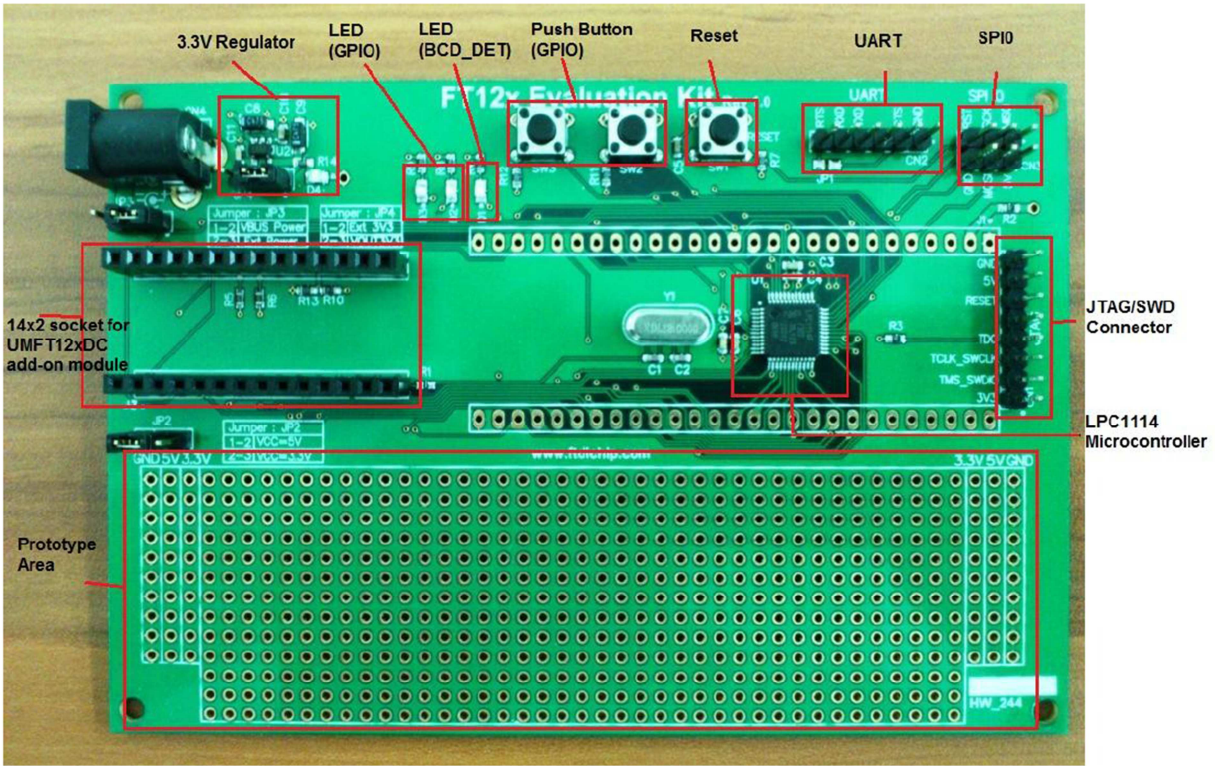


Figura 2.3.1 Placa principal UMFT12XEV [6]

La figura anterior muestra el diseño de la PCB con la posición de los componentes principales. Estos componentes incluyen:

Zócalo de 14x2 para el módulo adicional. Para insertar cualquiera de las tres placas hijas (UMFT120DC, UMFT121DC, UMFT122DC).

Regulador de 3.3V. Tiene una entrada de 5V del Vbus USB (alimentación por bus) o DC jack CN4 (alimentación propia) y una salida de 3.3V para los circuitos de la placa.

LED (GPIO). 2 diodos LED para indicar el estado de la GPIO. Puede ser utilizado para el estado del teclado HID.

LED (BCD_DET). Este LED indica si un cargador USB está conectado y detectado por FT121/FT122.

Pulsador (GPIO). 2 pulsadores para la entrada de control GPIO. Puede ser usado para las teclas de función del teclado HID.

Botón de reinicio. Botón de reinicio de hardware.

Conector UART. Este conector de 6 pines UARTG se puede utilizar para descargar el firmware en el sistema, así como puerto de depuración del firmware.

Conector SPIO. Este conector de 6 pines SPI se puede utilizar para conectarse a una placa esclava SPI.

Conector JTAG/SWD. Este conector de 8 pines es para la conexión a LCP-link para descargar el firmware y proceder a la depuración.

El microcontrolador LPC1114. 44 pines LPC1114 Cortex-M0 MCU para controlar el FT12x y otros periféricos.

Zona de prototipo. Para añadir circuitos de aplicación adicionales para el propósito de creación de prototipos.

2.3.1.2 Placa hija UMFT120DC

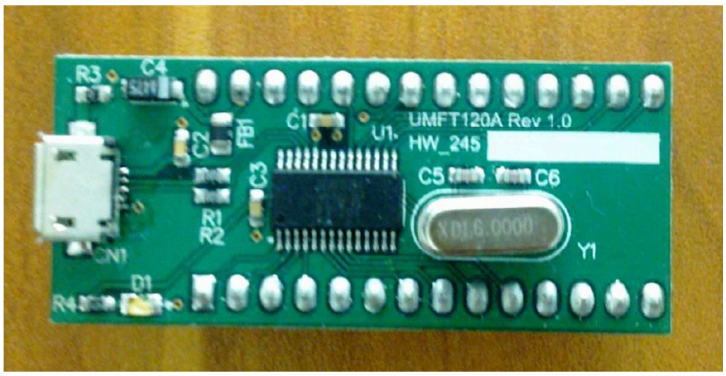


Figura 2.3.2 Placa hija UMFT120DC [6]

En la imagen anterior podemos ver la UMFT120DC. A continuación se enumeran y explican cada una de sus conexiones:

Nombre	Tipo	Descripción
DATA	IO	Bus de datos de 8 bits
ALE	I	Habilita la multiplexación de direcciones/datos. Activo a nivel alto
CS_n	I	Selección de chip. Activo a nivel bajo.
CLKOUT	O	Reloj de salida programable
RD_n	I	Habilita la lectura. Activo a nivel bajo.
WR_n	I	Habilita la escritura. Activo a nivel bajo.
DMREQ	O	Petición DMA
DMACK_n	I	Confirmación DMA. Activo a nivel bajo.
EOT_n	I	Fin de la transferencia DMA. Activo a nivel bajo. También funcionará como entrada de detección Vbus para aplicaciones con alimentación propia.
GL_n	OD	Indicador de actividad del bus USB. Activo a nivel bajo.
A0	I	Indica comandos o datos cuando no está activa la multiplexación. A0=1 indica comandos y A0=0 indica datos. Debe valer 1 cuando esté activa la multiplezación.
SUSPEND	I,OD	Suspensión y activación del dispositivo.
INT_n	OD	Interrupción. Activa a nivel bajo.
RESET_n	I	Reinicio asíncrono. Activo a nivel bajo.

Tabla 2.3.1 Descripción de las conexiones del UMFT120DC [5]

2.4 FPGA

Una FPGA (Field Programmable Gate Arrays) [7] [8] es un dispositivo semiconductor que se basa en una matriz de bloques lógicos configurables (CLBs) conectados mediante interconexiones programables. Las FPGAs pueden ser reprogramadas según los requisitos de las aplicaciones o funcionalidades deseadas tras su fabricación. Esta característica es la que distingue a las FPGAs de los circuitos integrados de aplicación específica (ASIC).

2.4.1 Arquitectura

Una FPGA está formada por bloques lógicos que se interconectan mediante canales de conexión vertical y horizontal a las celdas de entrada/salida. Se puede observar en la siguiente figura 2.4.1.

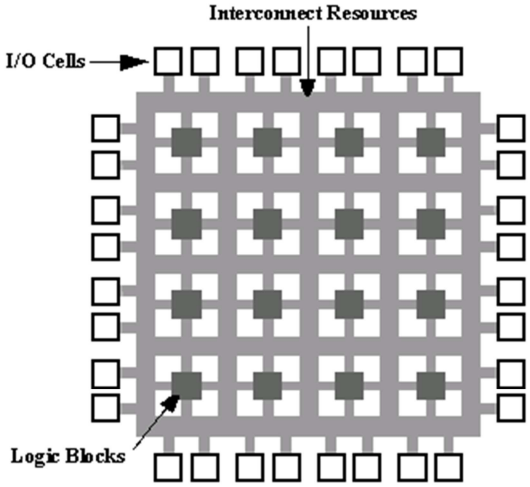


Figura 2.4.1 Arquitectura básica de una FPGAn [9]

Para la implementación de este proyecto se ha utilizado una FPGA de la familia Spartan 3E, la cual no posee una interfaz USB y se compone de cinco elementos funcionales programables fundamentales:

Bloques lógicos configurables (CLBs): contienen tablas de consulta (LUTs) flexibles que implementan elementos lógicos y de almacenamiento usados como *flip-flops* o *latches*. CLBs realizan una alta variedad de funciones lógicas así como almacenar datos.

Bloques de entrada/salida (IOBS): controlan el flujo de datos entre los pines de entrada/salida y la lógica interna del dispositivo. Cada IOB soporta flujo de datos bidireccionales además de operaciones triestado. Soporta una variedad de estándares de señales, incluyendo 4 estándares diferenciados de alto rendimiento. Registros Double Data-Rate (DDR) están incluidos.

Bloque RAM: proporciona almacenamiento de datos en forma de bloques de dos puertos de 18 Kbit.

Bloques Multiplicadores: aceptan dos números binarios de 18 bits como entrada y calculan el producto.

Bloque gestor de reloj digital (DCM): proporcionan calibración automática, soluciones totalmente digitales para la difusión, retrasos, multiplicaciones, divisiones y señales de reloj con desplazamiento de fase.

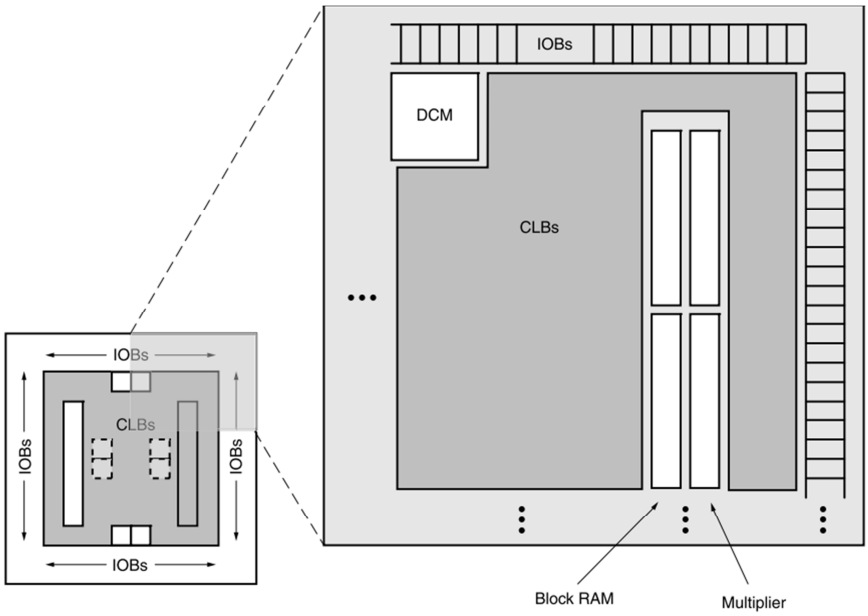


Figura 2.4.2 Arquitectura de la familia Spartan-3E [7]

3. Diseño

3.1 Sistema completo

3.1.1 Interfaz entrada/salida

Para la implementación de este proyecto se han utilizado dos elementos principales, una FPGA Spartan 3E y un FT120. A continuación se procederá a explicar cada uno de ellos y las Entradas/Salidas necesarias para esta implementación.

3.1.1.1 FPGA

La FPGA es un dispositivo lógico programable, que contiene bloques lógicos cuyas interconexiones y funcionalidad se pueden configurar utilizando un lenguaje de descripción hardware especializado.

La FPGA que se ha utilizado en este proyecto es una Spartan-3E de Xilinx, y se ha programado usando el lenguaje de descripción hardware VHDL.

Las entradas y salidas definidas en el código VHDL del sistema implementado se muestran en la figura 3.1.1 y se describen a continuación.

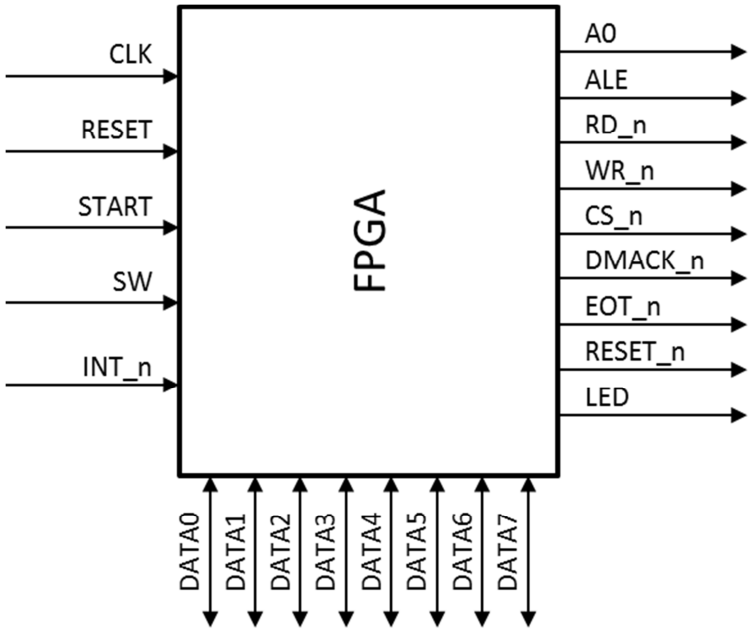


Figura 3.1.1 Interfaz entrada/salida FPGA

Entradas:

- CLK: Señal de reloj de 50MHz. Se utiliza el reloj que proporciona la propia FPGA.
- RESET: Señal que indica el reinicio del sistema. El sistema vuelve a sus condiciones iniciales. Se introduce mediante un botón de la FPGA.
- START: Señal que habilita el comienzo del programa. Se implementa con un botón de la FPGA.

- SW: Interruptores para seleccionar las señales a mostrar por los leds (4 bits).
- INT_n: Señal de interrupción generada por el FT120.

Salidas:

- A0: Señal que indica si la información transmitida son comandos o datos.
- ALE: Señal que habilita la multiplexación de direcciones y datos. En este caso permanecerá siempre deshabilitado.
- RD_n: Señal que habilita al FT120 para leer el bus de datos.
- WR_n: Señal que habilita al FT120 para escribir en el bus de datos.
- CS_n: Señal que selecciona el dispositivo para su uso.
- DMACK_n: Asentimiento DMA. No se usa DMA y está siempre desactivado.
- EOT_n: Fin de transferencia DMA. No se usa DMA y permanece desactivado.
- RESET_n: Señal que reinicia el dispositivo. Se le asigna el valor de la entrada RESET.
- LED: Señal que muestra los valores de diferentes señales. Se usan los leds que proporciona la FPGA (8 bits).

Entrada-Salida:

- DATA: Es el bus de Datos que utiliza la FPGA para comunicarse con el dispositivo. Está compuesto de ocho entradas-salidas.

Estas entradas y salidas se encuentran ubicadas en la FPGA como se muestra en la siguiente figura 3.1.2.

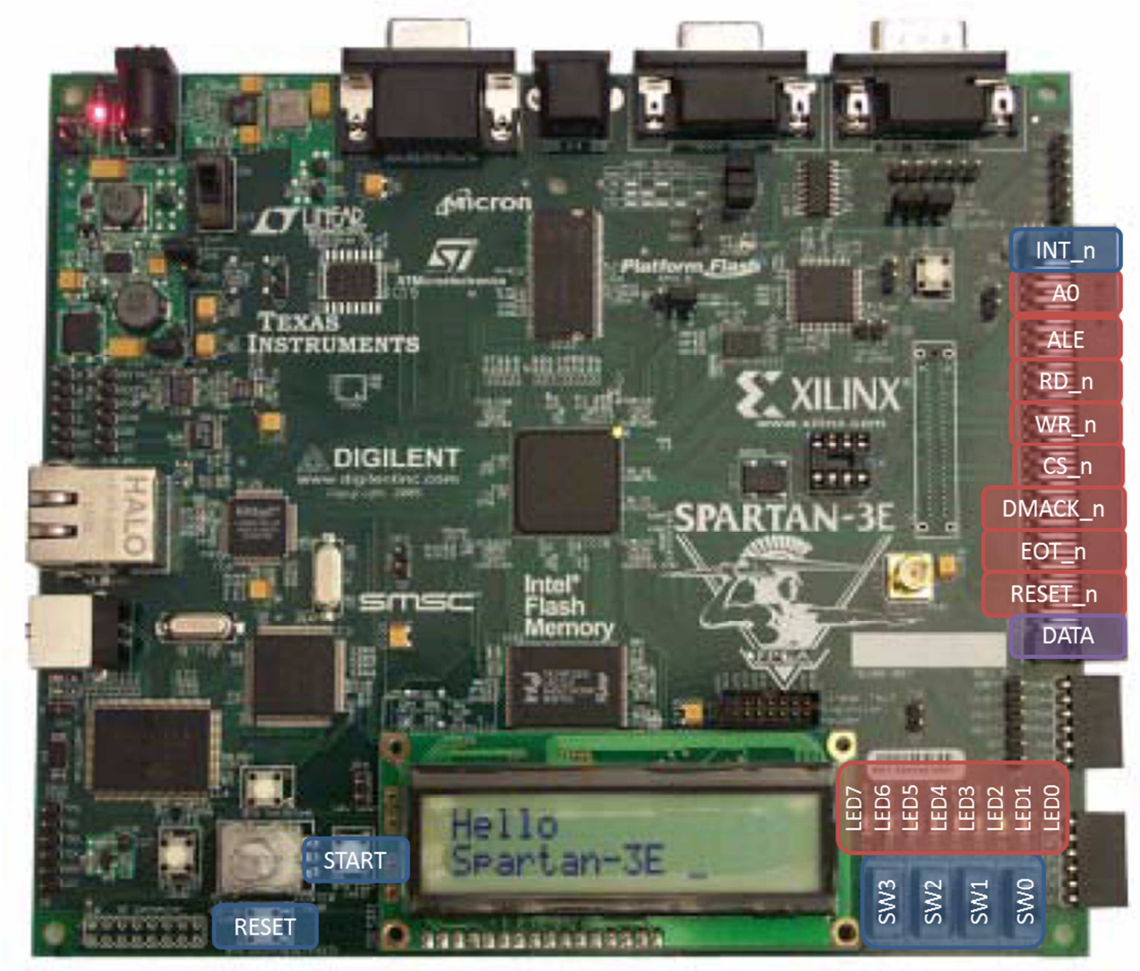


Figura 3.1.2 Entradas/Salidas de la FPGA

3.1.1.2 FT120

El FT120 es un controlador USB que sirve de intermediario y permite la comunicación del host USB del PC con la FPGA.

Las entradas y salidas del dispositivo [6] se muestran en la siguiente figura 3.1.3, correspondiéndose en gran parte a las explicadas en el apartado anterior 3.1.1.1. El resto de señales se detallarán a continuación.

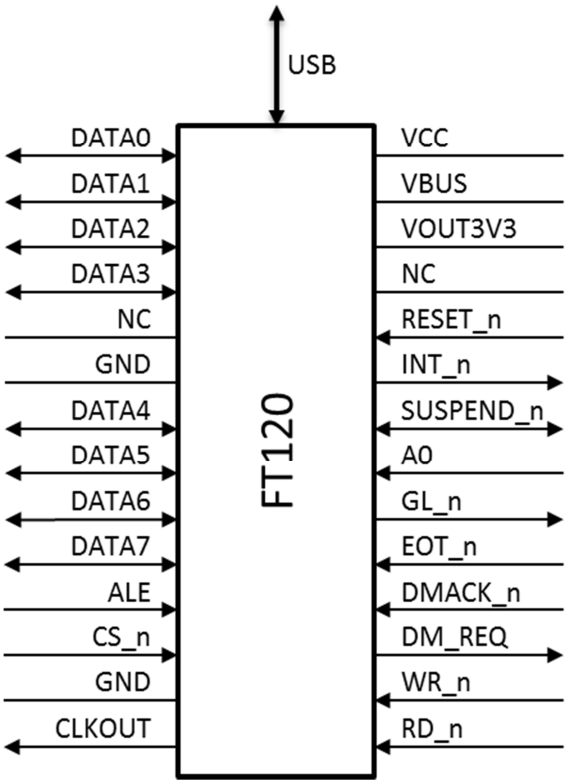


Figura 3.1.3 Entradas/Salidas del FT120

Salidas:

- CLKOUT: Salida de la señal de reloj. No ha sido utilizada en esta implementación.
- DM_REQ: Solicitud DMA. No se requiere, ya que no se implementa el DMA.
- GL_n: Indicador de la actividad del bus USB.
- INT_n: Debe ser conectada al igual que SUSPEND_n a una resistencia de 4.7 KΩ.

Entrada-Salida:

- SUSPEND_n: Suspende o activa el dispositivo. Lo conectamos a la fuente de 3.3V a través de una resistencia de 4.7 KΩ, como se indica en la especificación.
- USB: Puerto de conexión USB.

Otros:

- NC: Corresponde a los pines que no deben ser conectados.
- GND: Son pines que deben conectarse a tierra.
- VOUT3V3: Salida del regulador a 3.3 V.
- VBUS: Voltaje de 5V del conector USB.
- VCC: Alimentación VCC.

3.1.2 Diagrama de bloques del sistema completo

En este diagrama se muestra como se interconectan los diferentes dispositivos que conforman el sistema implementado.

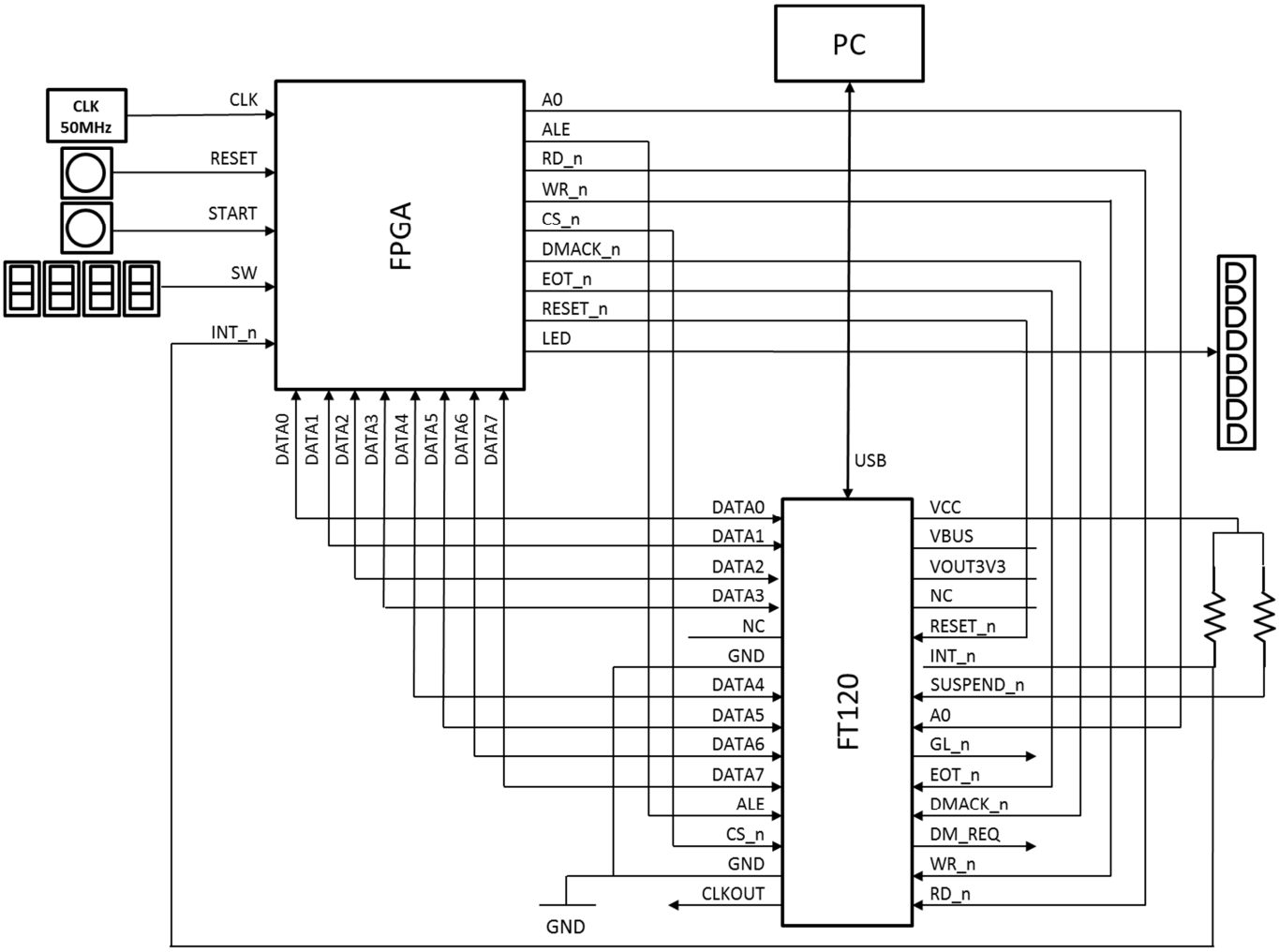


Figura 3.1.4 Diagrama de bloques

3.1.2.1 FPGA

El sistema grabado en la FPGA está compuesto por diversos bloques que se muestran en la siguiente figura.

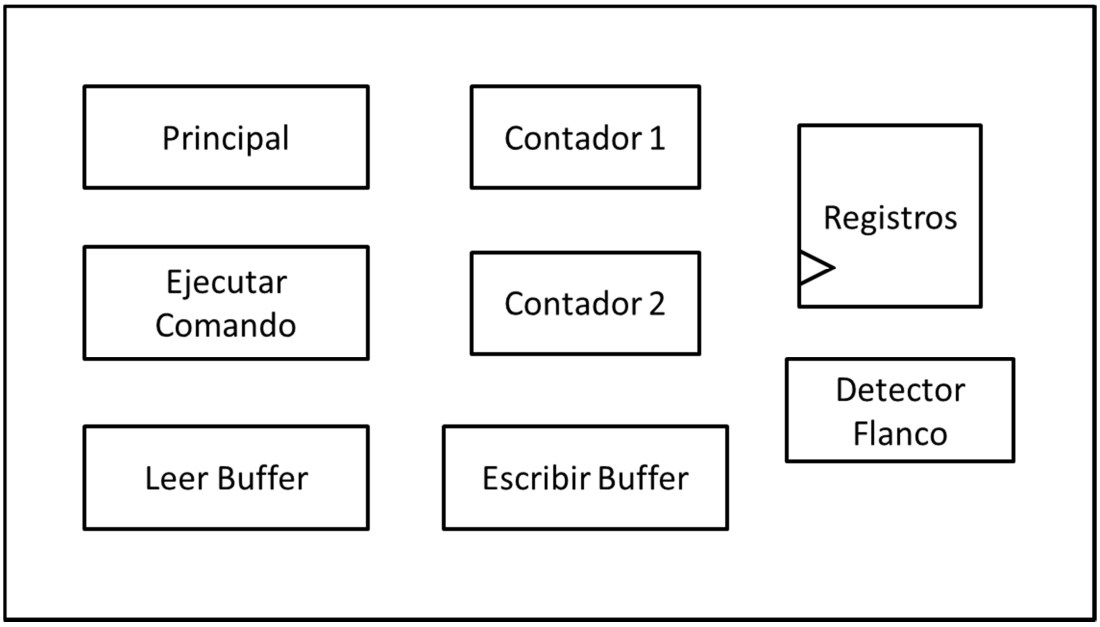


Figura 3.1.5 Bloques en la FPGA

- **Principal:** Es una máquina de estados que se encarga de tomar las decisiones sobre la evolución de la comunicación USB. En función de los datos recibidos desde el FT120, se le enviarán los comandos y datos que correspondan.

- **Ejecutar Comando:** Es una máquina de estados que se encarga de enviar los comandos y enviar o recibir los datos que correspondan según decida la máquina de estados Principal.

- **Leer *Buffer*:** Es una máquina de estados que se encarga de realizar una lectura del *buffer*, compuesta por tantas lecturas sucesivas como corresponda hasta recibir la carga útil del paquete.

- **Escribir *Buffer*:** Es una máquina de estados que se encarga de realizar una escritura del *buffer*, compuesta por tantas escrituras sucesivas como corresponda hasta enviar toda la información correspondiente.

- **Contador 1:** Es un contador que permite contabilizar el número de ciclos de espera que corresponda entre diferentes estados.

- **Contador 2:** Es un contador que permite contabilizar el número de lecturas o escrituras que se llevan a cabo hasta completar la lectura o escritura del *buffer*.

- **Detector de Flanco:** Este es un componente que se encarga de generar una señal de un pulso cuando la señal correspondiente de entrada se active.

- **Registros:** Este es un bloque síncrono encargado de dar valor a todas las señales de los biestables.

La jerarquía de las máquinas de estados diseñadas se muestra en la siguiente figura 3.1.6.

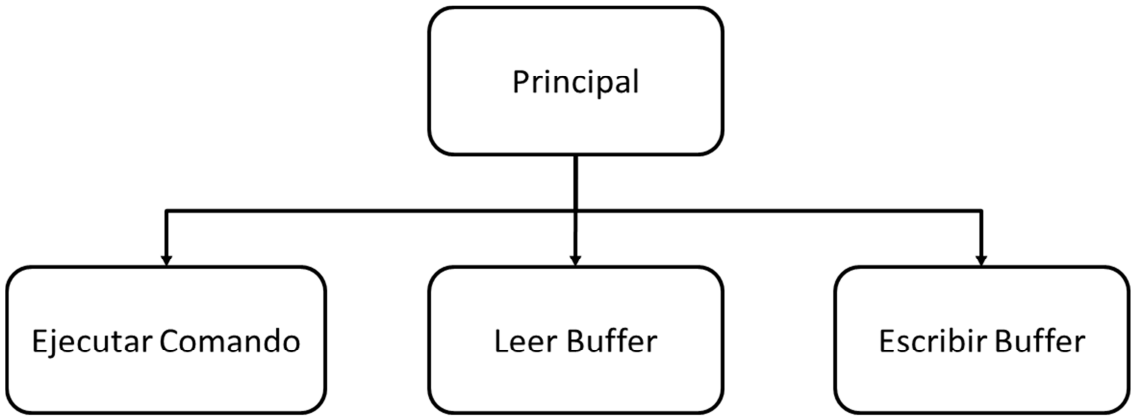


Figura 3.1.6 Árbol de las máquinas de estado

3.2 Máquina de Estados Principal

La máquina de estados principal es la encargada de la implementación del protocolo USB, es decir, se encarga de la comunicación entre la FPGA y el FT120. En ella, se deciden los comandos y datos que deben ser enviados al dispositivo en cada momento, teniendo en cuenta la información que proporcionan los datos recibidos de dicho dispositivo.

A continuación se procede a explicar la interfaz y descripción detallada de la máquina de estados Principal.

3.2.1 Interfaz entrada/salida

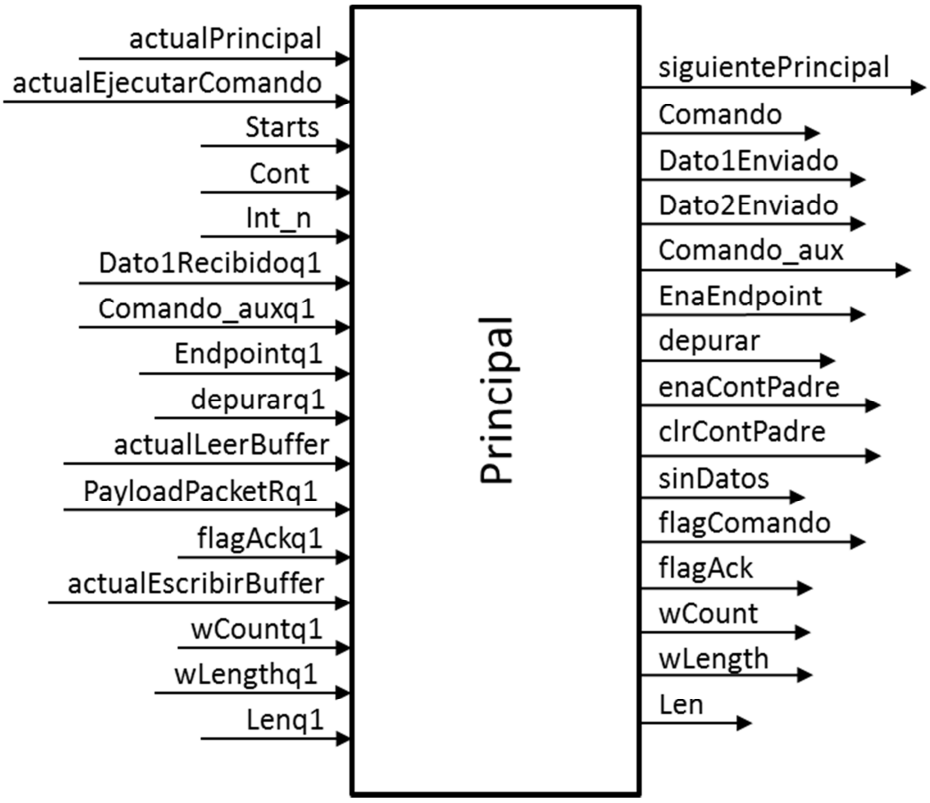


Figura 3.2.1 Interfaz de la máquina de estados Principal

Entradas:

- ActualPrincipal: Señal que indica el estado actual de la máquina de estados Principal.
- ActualEjecutarComando: Señal que indica el estado actual de la máquina de estados Ejecutar Comando.
- Starts: Señal que habilita el comienzo del programa.
- Cont: Señal de tipo entero con rango de 0 a 5000000. Almacena el valor del contador de ciclos de reloj para realizar una determinada espera. El valor máximo es 5000000 que corresponde a una espera de 100 ms.
- Int_n: Señal de interrupción que comunica que el FT120 tiene datos pendientes. Activa a nivel bajo.
- Dato1Recibidoq1: Señal de 8 bits para registrar el valor leído del bus de datos.

- Comando_auxq1: Señal de 8 bits que registra el valor del siguiente comando, que depende del *endpoint* seleccionado.
- Endpointq1: Señal de 8 bits para registrar el valor del *endpoint* seleccionado.
- Depurarq1: Señal de 8 bits para registrar un valor que indica el punto de ejecución en el que se encuentra el programa.
- ActualLeerBuffer: Señal que indica el estado actual de la máquina de estados Leer *Buffer*.
- PayloadPacketRq1: Señal de 8 bits para registrar la carga de los paquetes leídos, en la lectura del *buffer*.
- flagAckq1: Señal que registra el *flag* que indica si se ha asentido con un *setup* la selección del *endpoint*.
- ActualEscribirBuffer: Señal que indica el estado actual de la máquina de estados Escribir *Buffer*.
- wCountq1: Señal de tipo entero para registrar el número de paquetes que se han enviado hasta el momento del total que se deben enviar en la escritura del *buffer*.
- wLengthq1: Señal de tipo entero para registrar el número de paquetes total que han de enviarse en la escritura del *buffer*.
- Lenq1: Señal de 8 bits para registrar el número de paquetes que faltan por enviar en la escritura del *buffer*.

Salidas:

- siguientePrincipal: Señal que indica el siguiente estado de la máquina de estados Principal.
- Comando: Señal de 8 bits que guarda el valor del comando que corresponda enviar.
- Dato1Enviado: Señal de 8 bits que guarda el primer dato enviado tras un determinado comando.
- Dato2Enviado: Señal de 8 bits que guarda el segundo dato a enviar en ciertos comandos.
- Comando_aux: Señal de 8 bits que contiene el valor del siguiente comando, que depende del *endpoint* seleccionado.
- EnaEndpoint: Señal que habilita la carga del *endpoint* seleccionado por el dispositivo en la señal *endpoint*.
- Depurar: Señal de 8 bits que indica el punto de ejecución en el que se encuentra el programa.
- EnaContPadre: Señal que habilita a la máquina Principal para usar el contador que se usa en las diferentes esperas del sistema.
- ClrContPadre: Señal de la máquina de estados padre que limpia el valor del contador de espera del sistema.
- SinDatos: Señal que indica sí tras un comando se realizan transferencias de datos.
- FlagComando: *Flag* cuyo valor indica si a continuación de un determinado comando se deben recibir (0) o enviar datos (1).
- FlagAck: Señal que indica si se ha asentido con un *setup* la selección del *endpoint*.
- wCount: Señal entera que contiene el número de paquetes que se han enviado hasta el momento del total que se han de enviar en la escritura del *buffer*.

- WLength: Señal entera que contiene el número de paquetes total que han de enviarse en la escritura del *buffer*.
- Len: Señal de 8 bits que guarda el número de paquetes que faltan por enviar en la escritura del *buffer*.

3.2.2 Descripción

En el Anexo 1 de este proyecto, se encuentra el diagrama de flujo completo de la máquina de estados Principal, que puede ayudar a la comprensión del comportamiento del sistema.

Debido al gran tamaño y complejidad de esta máquina de estados, se ha procedido a dividirla en 5 partes y analizar cada una de ellas por separado. Estas partes son: inicialización y configuración del chip, manejador de interrupciones del FT120 y *endpoints* de control, recepción del paquete *Setup*, atender solicitud *Get Descriptor* y escribir más datos.

A continuación se muestra un diagrama que relaciona las partes de las que se compone la máquina de estados, estando cada una de ellas formada por diferentes estados.

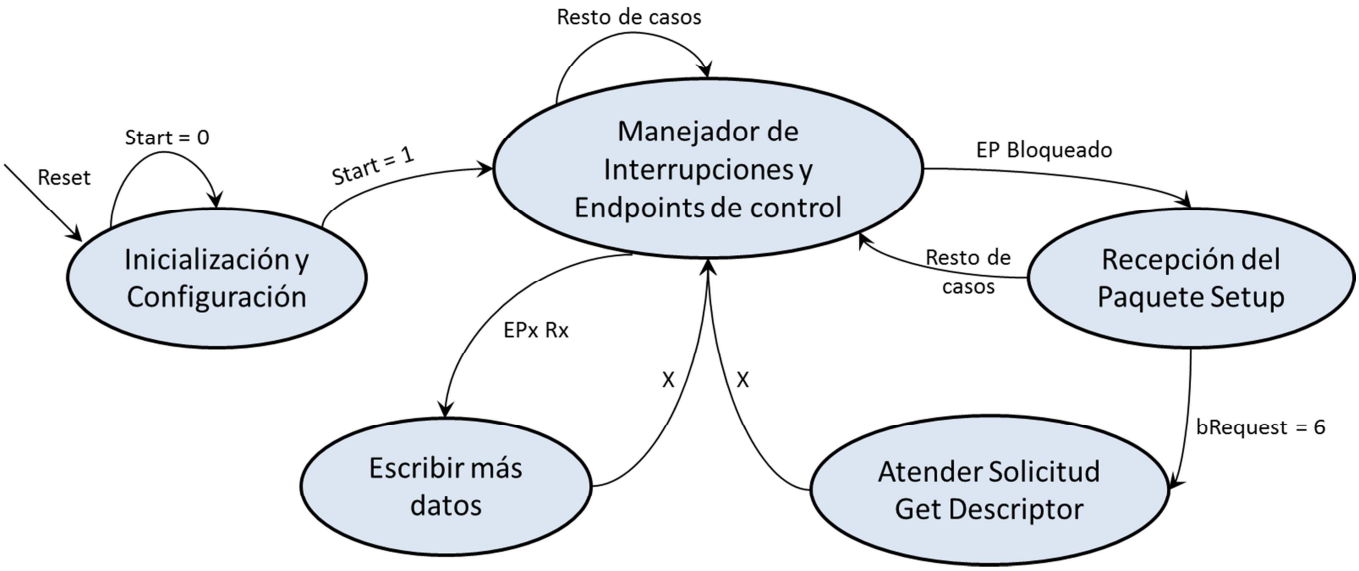


Figura 3.2.3 División en 5 partes de la máquina de estados Principal

Para comprender la funcionalidad de cada una de las partes, se procede a explicar en detalle cada una de ellas y las conexiones que las relacionan.

3.2.2.1 Inicialización y configuración del Chip

En primer lugar se muestra el diagrama de flujo que corresponde a esta etapa de la máquina de estados Principal.

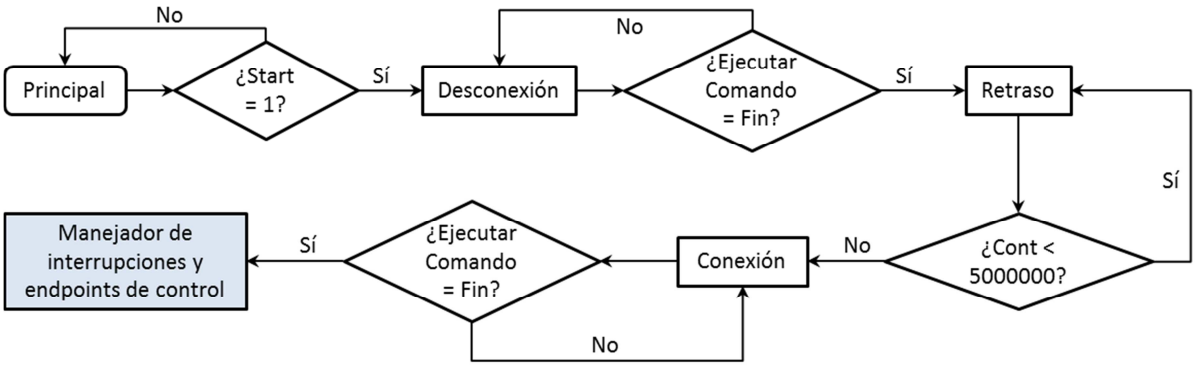


Figura 3.2.4 Diagrama de flujo de inicialización y configuración

El FT120 debe ser inicializado [10] antes de que comience la comunicación a través del bus USB. Esto es lo que se lleva a cabo en los estados iniciales de la máquina de estados principal, que se muestran en la siguiente figura 3.2.5.

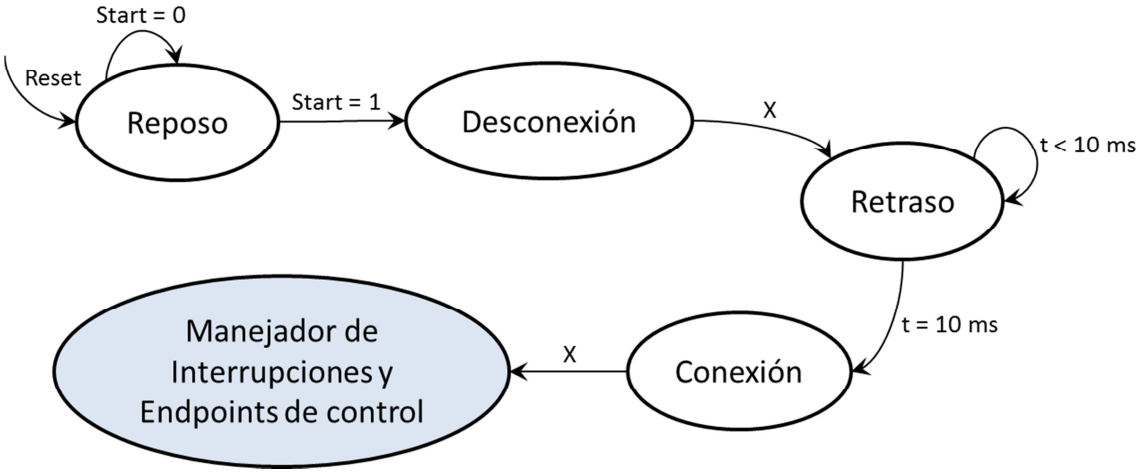


Figura 3.2.5 Inicialización y configuración de la máquina de estados Principal

La máquina de estados principal parte del estado de Reposo tras recibir un *reset*, en el cual permanecerá hasta que se active la señal de entrada Start. Cuando esto se produce, se pasa al estado de Desconexión.

En el estado Desconexión, se envían los comandos y datos necesarios para la desconexión del dispositivo. Aunque a priori el dispositivo se encuentra desconectado, esta es la manera de asegurarse de que esto es así. El comando utilizado es *Set Mode*, que se encarga de establecer la configuración inicial del dispositivo. A continuación se envían dos bytes de datos correspondientes a dicho comando. El primer dato proporciona varios parámetros de configuración del chip, para el estado desconectado: CLKOUT no cambia durante la suspensión USB, el reloj interno se para durante la suspensión, las transacciones NAK y ERROR no generan interrupción, se desactiva la resistencia de *pull-up* en el pin USBDP y se configura el modo 00 del EP2 (*Bulk/Interrupt*). El segundo byte de datos establece el factor de división del reloj (CDF) que determina la frecuencia de salida del reloj en el pin CLKOUT. La frecuencia se calcula como *Frequency* = 48 MHz / (CDF +1), por lo que se fija el CDF = 3 para obtener una frecuencia de 12MHz. Al finalizar estas transacciones, se pasará al siguiente estado Retraso.

En el estado Retraso se procede a realizar una espera de 100 milisegundos. Para ello se utiliza un contador de 5.000.000, ya que el ciclo de reloj de nuestro sistema es de 20 nanosegundos (la frecuencia de reloj es 50 MHz). Nº ciclos = 100 ms / 20 ns = 5.000.000. Esta espera se debe al reseteo del dispositivo que realiza el *hub* cuando el *host* aprende un nuevo dispositivo. La condición de reinicio dura al menos 100 milisegundos. A continuación se pasará al estado Conexión.

En el estado Conexión, se envían los comandos y datos necesarios para la configuración inicial del dispositivo. El comando utilizado vuelve a ser *Set Mode*, pero en este caso los parámetros de configuración corresponderán al estado conectado del dispositivo. El primer byte de datos establece la siguiente configuración: CLKOUT conmuta a 30KHz durante la suspensión USB, el reloj interno se para durante la suspensión, las transacciones NAK y ERROR no generan interrupción, la resistencia de *pull-up* en el pin USBDP se habilita cuando está presente Vbus y se configura el modo 00 del EP2 (*Bulk/Interrupt*). El segundo byte de datos configura el CDF igual que en el estado Desconexión, CDF=3 para obtener una frecuencia de 12MHz. Con esto finaliza la inicialización del dispositivo y se pasa a la siguiente fase, manejador de interrupciones y *endpoints* de control.

Se debe tener en cuenta que además durante esta etapa se configuran los *endpoints* 0 y 1 como *endpoints* de control, con un tamaño máximo de *buffer* de 16 bytes. Estos se habilitan por defecto y no se pueden modificar.

3.2.2.2 Manejador de interrupciones del FT120 y *Endpoints* de control

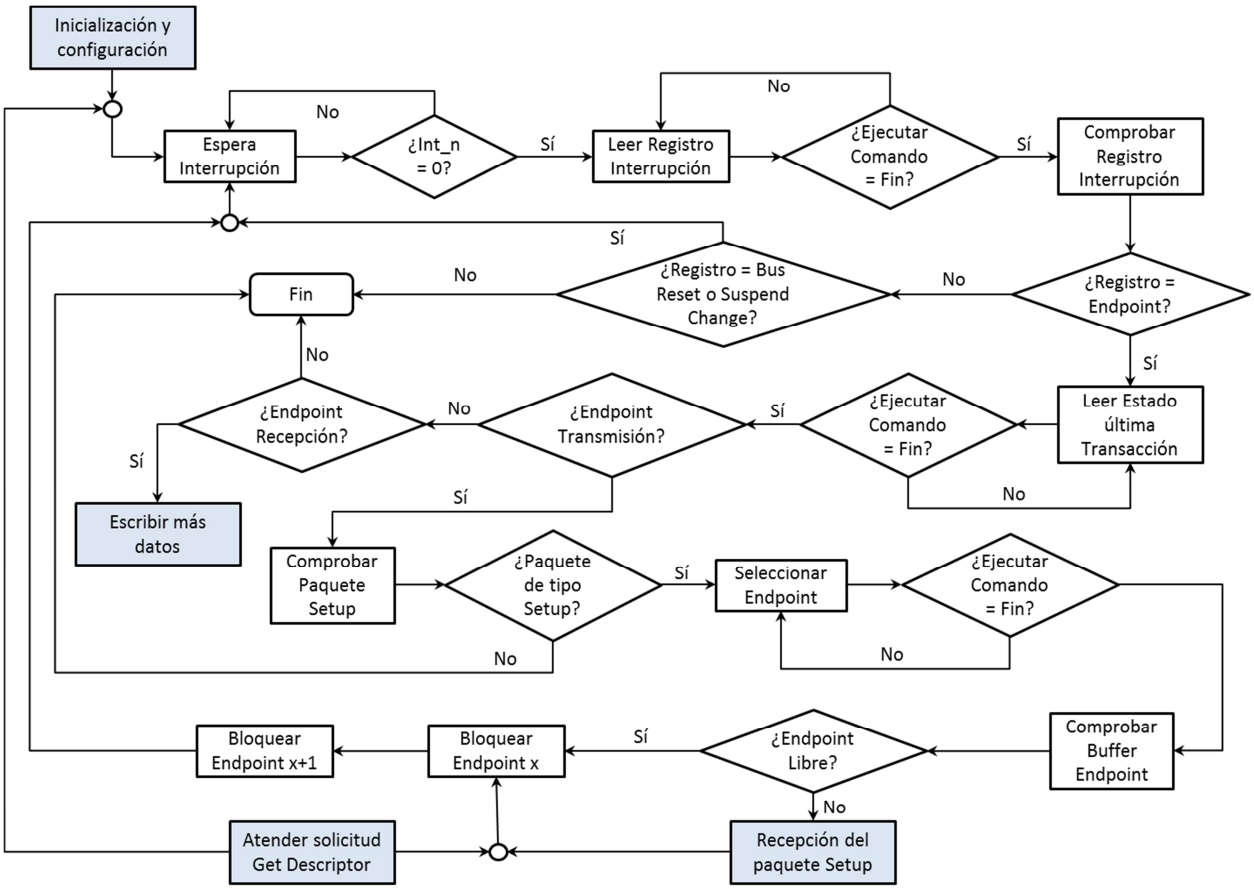


Figura 3.2.6 Diagrama de flujo del manejador de interrupciones del FT120 y *endpoints* de control

En esta etapa de la máquina de estados principal, se lleva a cabo la atención a las interrupciones generadas por el FT120 al recibir un paquete. Cuando se trata de un paquete de *setup* se procede a bloquear el *endpoint* de control a utilizar. A continuación se muestra la parte de la máquina de estados que corresponde a esta etapa, Figura 3.2.7.

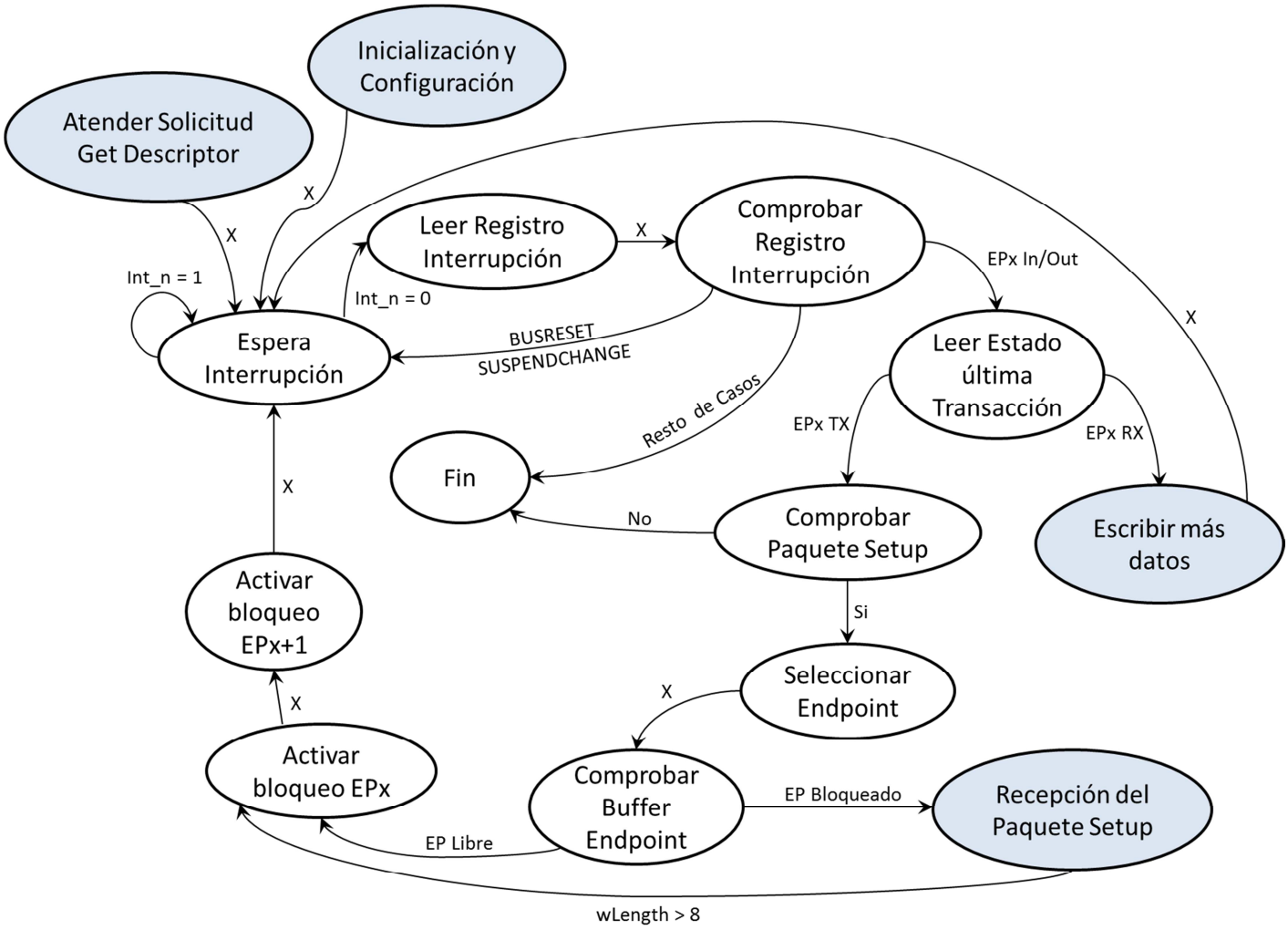


Figura 3.2.7 Manejador de interrupciones y *endpoints* de control de la máquina Principal

A esta etapa se llega cuando se completa la inicialización y configuración del dispositivo o se termina de atender una solicitud, tal y como se muestra en la anterior figura.

Esta fase comienza en el estado Espera Interrupción, en el cual permanece la máquina de estados principal hasta que se activa la salida de interrupción (Int_n) del FT120. Esta interrupción se activará cuando el FT120 reciba un paquete USB, que inicialmente debe ser de tipo *setup*. Esta interrupción es activa a nivel bajo y provocará el paso al estado Leer Registro Interrupción.

En el estado Leer Registro Interrupción, se procede a enviar el comando *Read Interrupt Register*, que sirve para solicitar al FT120 que le envíe a la máquina de estados la información que contiene el registro de interrupción. Durante este estado se procederá también a la lectura de dicha respuesta y se pasará al siguiente estado Comprobar Registro Interrupción.

En el estado Comprobar Registro Interrupción, se comprueba el contenido de dicho registro, que indica el *buffer* al que va dirigida la interrupción. Los posibles *buffers* son el de entrada o salida de cada uno de los *endpoints*: *Endpoint 0 Out (0), Endpoint 0 In (1), Endpoint 1 Out (2), Endpoint 1 In (3), Endpoint 2 Out(4)* o *Endpoint 2 In(5)*. A parte de los casos de transferencia de datos anteriores, el FT120 también genera interrupciones cuando el *host* suspende el bus, o se lleva a cabo un reseteo del bus. En estos dos últimos casos, la máquina de estados volverá al estado Espera Interrupción. En el caso de que se seleccione uno de los posibles *buffer*, se guardará dicho valor y se pasará al estado Leer Estado última Transacción. Para el resto de valores del registro de interrupción, se considerará que ha ocurrido un error y se terminará con el paso al estado Fin.

En el estado Leer Estado última Transacción, se envía el comando correspondiente *Read Last Transaction Status.* Para el caso de que se hayan seleccionado los *buffers* de salida (TX) de los *endpoints*, se comprobará el tipo de paquete de la última transacción. Esta información se leerá en el byte de datos con el que responde el dispositivo. En este caso, después se pasará al estado Comprobar Paquete *Setup*. Si por el contrario, se han seleccionado los *buffer* de entrada (RX) de los *endpoints*, este comando únicamente servirá para limpiar el *flag* de interrupción. En este caso, significa que el *host* está esperando a recibir más datos y la siguiente etapa será Escribir más Datos.

En el estado Comprobar Paquete *Setup*, se procede a analizar el byte de datos devuelto en el estado anterior. En concreto se comprueba el valor del bit 5, que indica si el paquete de la última transacción es de tipo *setup* o no. En el caso de que no lo sea, se finaliza pasando al estado Fin. Si se trata de un paquete *setup*, se pasa al estado Seleccionar *Endpoint*.

En el estado Seleccionar *Endpoint*, se envía el comando *Select Endpoint*, que corresponde al número de *endpoint* seleccionado anteriormente (0, 1, 2, 3, 4 o 5). Después se realiza la lectura del byte de datos con la que responde el FT120. A continuación se pasa al estado Comprobar *Buffer Endpoint*.

En el estado Comprobar *Buffer Endpoint*, se comprueba el byte de datos con el que responde el dispositivo al seleccionar el *endpoint.* Si el bit 0 es un 0, indicará que el *endpoint* está libre, y se pasará al estado Activar Bloqueo EPx. Si el bit 0 es un 1, indicará que el *endpoint* se encuentra ya bloqueado (se bloqueó el *endpoint* anteriormente), y el siguiente estado será de la etapa Recepción del paquete *Setup*.

En el estado Activar Bloqueo EPx, se procede a bloquear el *endpoint* para el uso de este dispositivo. Para ello se envía el comando Set *Endpoint* Status, en el que se indica el número de *endpoint*. También se envía un byte de datos de valor 01h que habilita el bloqueo del *endpoint*. Después se pasa al siguiente estado Activar Bloqueo EPx+1.

En el estado Activar Bloqueo EPx+1, se procede igual que en el estado anterior, con la diferencia de que el *endpoint* bloqueado se corresponde al siguiente *endpoint* consecutivo. Esto es así ya que se bloquea el *endpoint* EPx (de salida) para la trasmisión de datos desde el dispositivo y el *endpoint* EPx+1 (de entrada) para la recepción de los datos en el dispositivo. El siguiente estado será siempre Espera Interrupción.

3.2.2.3 Recepción del paquete *Setup*

A continuación se muestra el diagrama de flujo que corresponde a esta fase de la máquina de estados Principal.

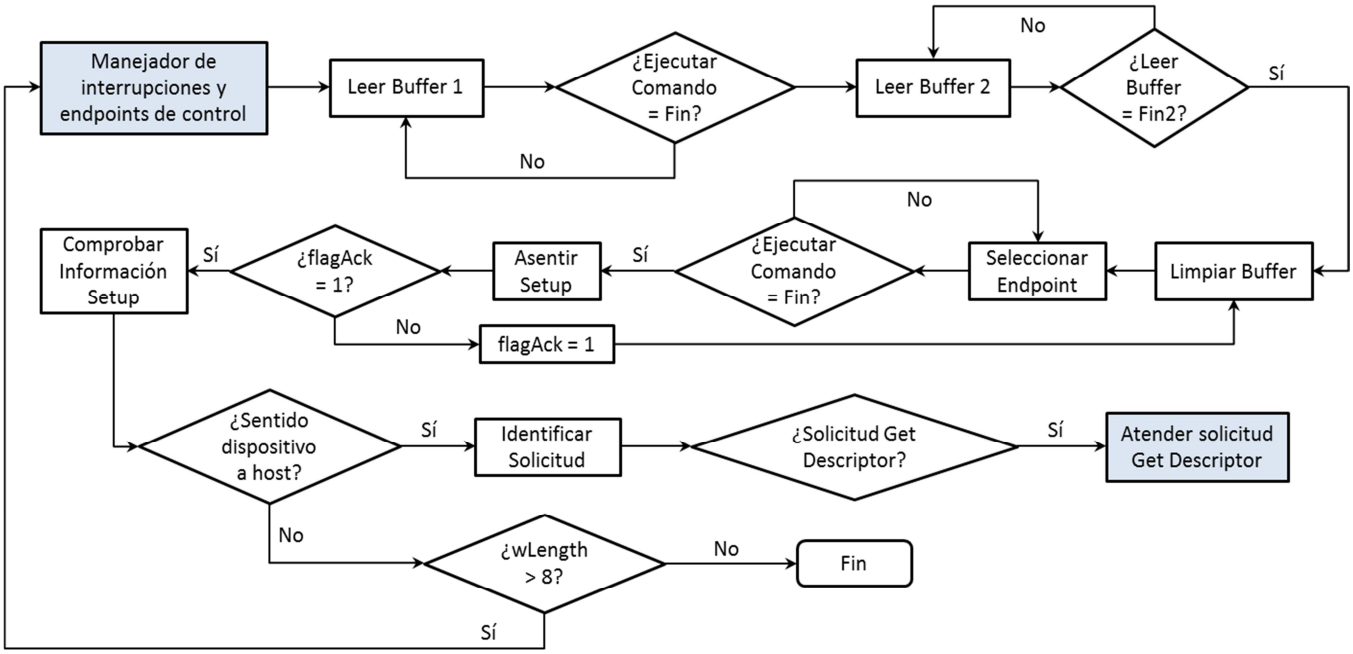


Figura 3.2.8 Diagrama de flujo de recepción del paquete setup

Una vez bloqueados los *buffers* de los *endpoints* correspondientes, se llega a esta etapa en la que el dispositivo FT120 envía la información recibida en el paquete *setup*. Se trata de 8 bytes de datos que tienen el siguiente formato:

Campo	Tamaño	Descripción
bmRequestType	1 byte	Especifica la dirección del flujo de datos, el tipo de la solicitud, y el receptor.
bRequest	1 byte	Especifica la solicitud del *host*, por ejemplo Get_Descriptor.
wValue	2 bytes	El *host* lo utiliza para pasar información al dispositivo, por ejemplo la dirección asignada.
wIndex	2 bytes	El *host* lo utiliza para pasar información al dispositivo, por ejemplo el número de interfaz o de *endpoint*.
wLength	2bytes	Número de bytes a transferir si hay fase de datos.

Tabla 3.2.1 Paquete *Setup*

Los estados que corresponden a esta fase de la máquina de estados principal, se muestran a continuación en la figura 3.2.9.

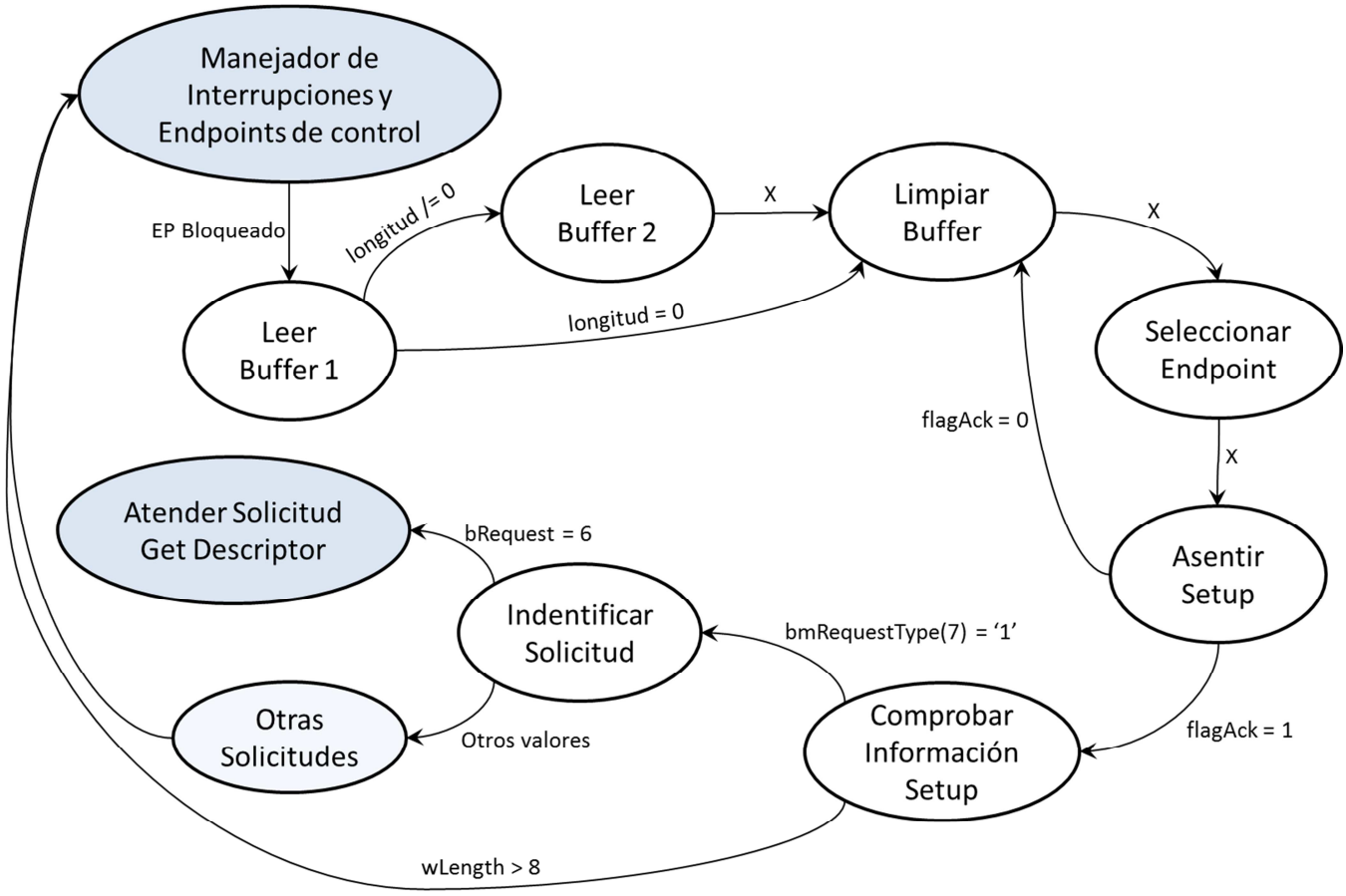


Figura 3.2.9 Recepción del paquete *setup* de la máquina de estados Principal

Cuando se encuentra bloqueado el *endpoint* seleccionado en la etapa manejador de interrupciones y *endpoints* de control, se llega a esta fase recepción del paquete *setup*.

Esta fase de la máquina de estados principal comienza en el estado Leer *Buffer* 1. En él se lleva a cabo el envío del comando *Read Buffer*, que sirve para leer el paquete recibido desde el *buffer* de salida del *endpoint* seleccionado anteriormente. A continuación, se realiza la lectura de los dos primeros bytes de datos. El primero no tiene información de interés, pero el segundo es el que indica la longitud de la carga útil del paquete, es decir, el número de lecturas de un byte que se deberán realizar a continuación. Si este valor es 0, el siguiente estado al que se pasará es Limpiar *Buffer*. Para el resto de valores, el siguiente estado será Leer *Buffer* 2.

En el estado Leer *Buffer* 2, se realizan el resto de lecturas necesarias hasta completar la lectura del paquete recibido en el *buffer* de salida del *endpoint* seleccionado. El número mínimo de lecturas que se llevarán a cabo son 8, ya que corresponde a la longitud del paquete de *setup* mostrado en la tabla 3.2.1. Tras obtener toda la información transferida por el dispositivo, se pasará al siguiente estado Limpiar *Buffer*.

En el estado Limpiar *Buffer*, se procede a enviar el comando correspondiente *Clear Buffer*. Este sirve para que se libere el *buffer* seleccionado, y así poder recibir el siguiente paquete enviado por el *host* USB. El siguiente estado será Seleccionar *Endpoint*.

En el estado Seleccionar *Endpoint*, se procede a seleccionar de nuevo el *buffer* de salida del *endpoint* limpiado anteriormente. Para ello se envía el comando *Select Endpoint*, y no se envían ni reciben datos. El siguiente estado será Asentir *Setup*.

En el estado Asentir *Setup*, se envía el comando *Acknowledge Setup*, que sirve para volver a habilitar los comandos *Clear Buffer* y *Validate Buffer*. Esto debe realizarse para el *endpoint* de entrada y de salida. En este estado se comprueba el valor del flagAck, que indica si el *endpoint* limpiado y habilitado es el de entrada o salida. Si es el de salida se modifica el valor del flagAck y se vuelve al estado Limpiar *Buffer*, que en este caso se ejecutará para el *endpoint* de entrada. En el caso de que sea el de entrada, indicará que ya están ambos *endpoints* listos y se pasará al estado Comprobar *Setup*.

En el estado Comprobar Información *Setup*, se analiza la información recibida en el paquete *setup*. Se comprueba si *bmRequestType(7)* es 1, que indica que el sentido de los datos es del dispositivo al *host*, por lo que se debe atender su solicitud y contestar. Para ello el siguiente estado será Identificar Solicitud. Sí el sentido es el contrario, se comprueba que la wLength es mayor que 8, y si es así se pasa al estado Activo Bloqueo EPx de la fase manejador de interrupciones y *endpoints* de control.

En el estado Identificar Solicitud, la primera comprobación que se hace es que *bmRequestType(5)* y *bmRequestType(6)* son iguales a 0. Esto significa que la solicitud recibida es de tipo *Standard*. A continuación, se procede a comprobar el valor del campo *bRequest*, que indica la solicitud que ha enviado el *host* USB. Por ejemplo, un 5 indica un *Set Address*, un 6 un *Get Descriptor* y un 9 un *Set Configuration*. En función de este valor se pasará al siguiente estado correspondiente. Como por ahora solo ha sido implementada la solicitud *Get Descriptor*, para el resto de casos se volverá a la fase manejador de interrupciones y *endpoints* de control.

3.2.2.4 Atender solicitud *Get Descriptor*

Primeramente, se puede observar el diagrama de flujo correspondiente en la siguiente figura.

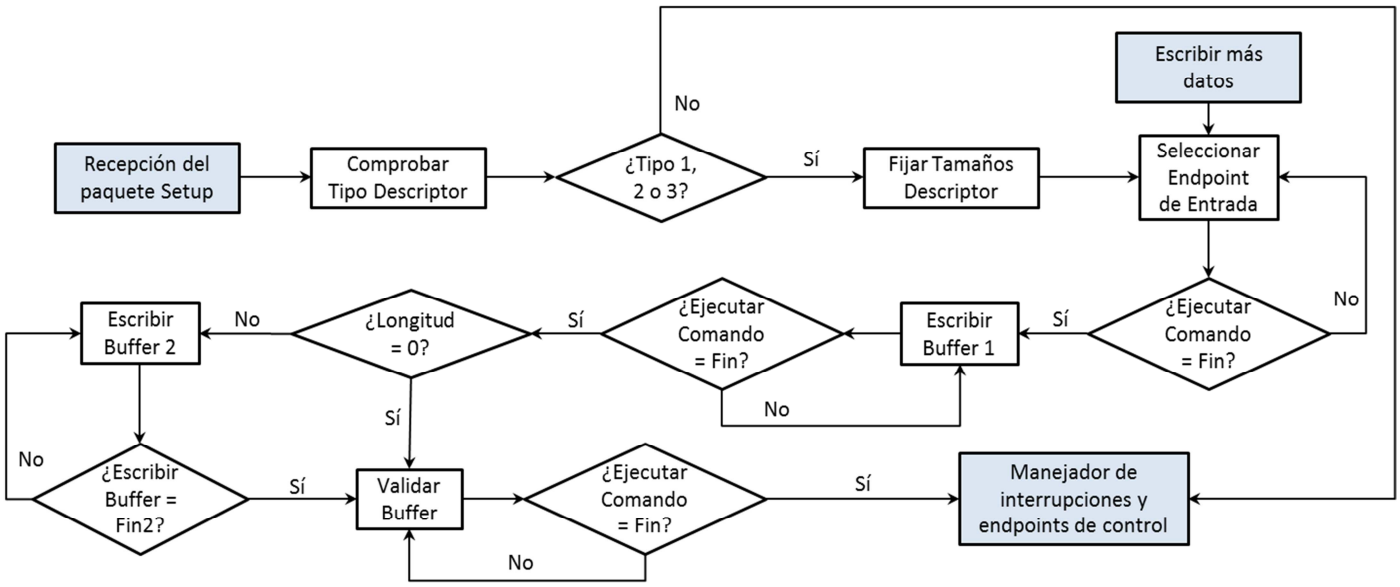


Figura 3.2.10 Diagrama de flujo de atender solicitud Get Descriptor

En esta etapa de la máquina de estados principal se procede a responder a la solicitud *Get Descriptor*, realizando una escritura en el *buffer* del descriptor solicitado.

En la siguiente figura 3.2.11 se pueden ver los estados que compondrían esta etapa.

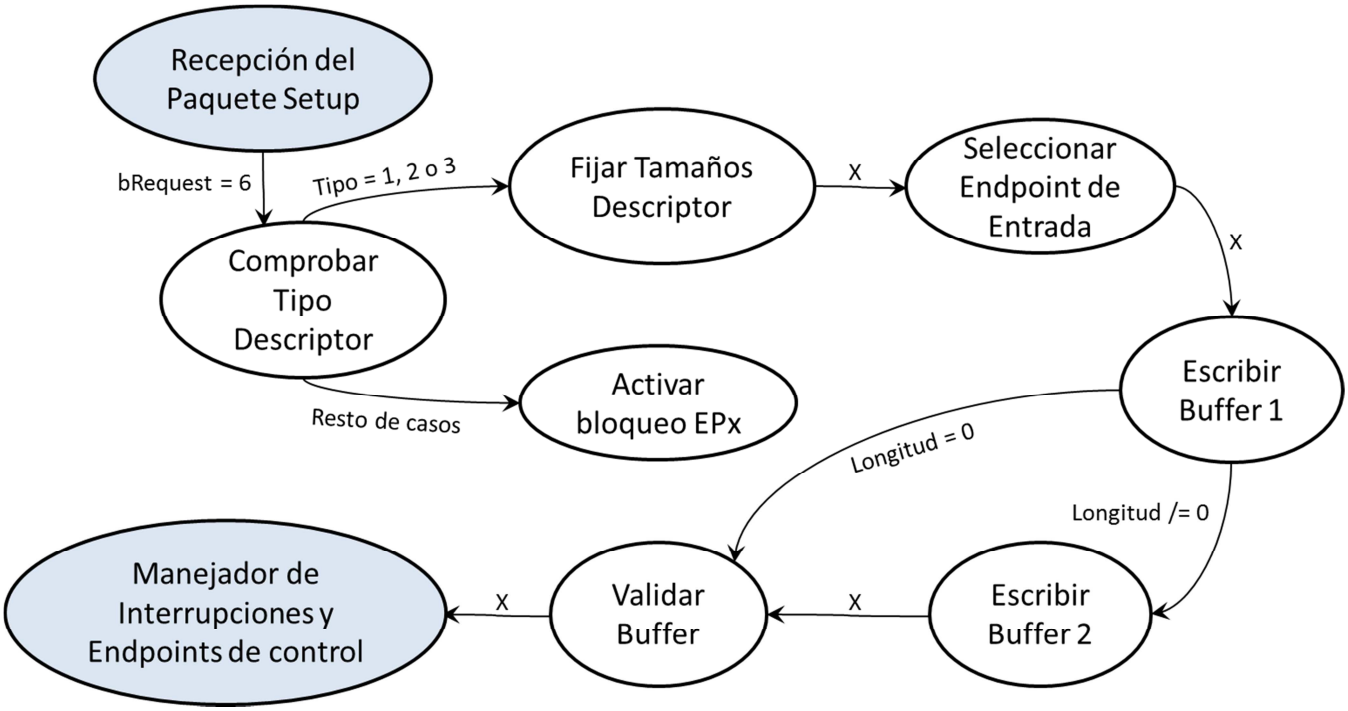


Figura 3.2.11 Atender solicitud *Get Descriptor* de la máquina Principal

A esta parte de la máquina de estados se llega cuando tras la fase recepción del paquete *setup*, se ha recibido una solicitud de tipo *Get Descriptor*.

El primer estado es Comprobar Tipo Descriptor, en el que se comprueba el tipo de descriptor solicitado por el *host* USB. Esta información está en el campo *wValue* del paquete *setup* recibido. Los posibles valores son 1 para el descriptor de dispositivo, 2 para el de configuración y 3 para el de cadena. Si el valor recibido fuera otro, se pasará al estado Activo Bloqueo EPx. La primera solicitud corresponderá al descriptor de dispositivo. En la siguiente tabla 3.2.2 se muestran los campos de los que se compone.

Campo	Tamaño	Descripción
bLength	1 byte	Tamaño del descriptor en bytes
bDescriptorType	1 byte	Constante descriptor de dispositivo (0x01)
bcdUSB	2 bytes	Número de versión de la especificación USB
bDeviceClass	1 byte	Código de clase
bDeviceSubclass	1 byte	Código de subclase
bDeviceProtocol	1 byte	Código de Protocolo
bMaxPacketSize0	1 byte	Tamaño máximo de paquete para el *endpoint* 0
idVendor	2 bytes	Identificación del vendedor
idProduct	2 bytes	Identificación del producto
bcdDevice	2 bytes	Número de versión del dispositivo (BCD)
iManufacturer	1 byte	Índice de descriptor de cadena para el fabricante
iProduct	1 byte	Índice de descriptor de cadena para el producto

iSerialNumber	1 byte	Índice de descriptor de cadena que contiene el número de serie
bNumConfigurations	1 byte	Número de posibles configuraciones

Tabla 3.2.2 Descriptor de dispositivo

El siguiente estado es Fijar Tamaño Descriptor, en el que se comprueba el valor de *wLength*, que es el campo del paquete *setup* que indica el número de bytes con los que se debe contestar a su solicitud. La longitud máxima de un descriptor es de 18 bytes, por lo que si el campo *wLength* es mayor, se limitará el número de bytes de la contestación a 18. En el caso de que *wLength* esté dentro del rango, se enviarán tantos bytes como ahí se indiquen. El siguiente estado será Seleccionar *Endpoint* de Entrada.

En el estado Seleccionar *Endpoint* de Entrada, se procede a seleccionar el *buffer* de entrada del *endpoint*, ya que en este caso se procederán a enviar datos al FT120, no ha recibirlos. Por ello se enviará el comando correspondiente *Select Endpoint*. El siguiente estado al que se pasa es Escribir *Buffer* 1.

En Escribir *Buffer* 1, se realiza el envío del comando correspondiente *Write Buffer*. Este sirve para indicarle al dispositivo que se va a escribir la respuesta a la solicitud en el *buffer* de entrada del *endpoint* seleccionado. Seguidamente se envían los dos primeros bytes de datos, que se organizan de la misma forma que para el comando *Read Buffer*. El primer byte contendrá siempre el valor 0x00, y el segundo byte la longitud de la carga útil del paquete, es decir, el número de escrituras de un byte que formarán la respuesta. Si la longitud es 0, el siguiente estado será Validar *Buffer*, para el resto de valores será Escribir *Buffer* 2.

En el estado Escribir *Buffer* 2, se realizan el resto de escrituras necesarias para enviar toda la información del descriptor solicitado. Esto solo es completamente cierto cuando el número de escrituras sea inferior al tamaño máximo de paquete del *endpoint*. Cuando es superior, se enviarán solo los 16 primeros bytes, que llenarán el *buffer*, y el resto se enviarán en la etapa Escribir más Datos. Esto sucede en el caso del descriptor de dispositivo, ya que su longitud son 18 bytes. Tanto si hemos conseguido enviar toda la información o no, el siguiente estado será Validar *Buffer*.

En Validar *Buffer* se envía el comando *Validate Buffer*, que sirve para indicar que la escritura en el *buffer* está completa. Cuando llega esta señal el FT120 sabe que el paquete puede ser enviado al *host* USB. El siguiente estado será Espera Interrupción de la etapa manejador de interrupciones y *endpoints* de control.

3.2.2.5 Escribir más datos

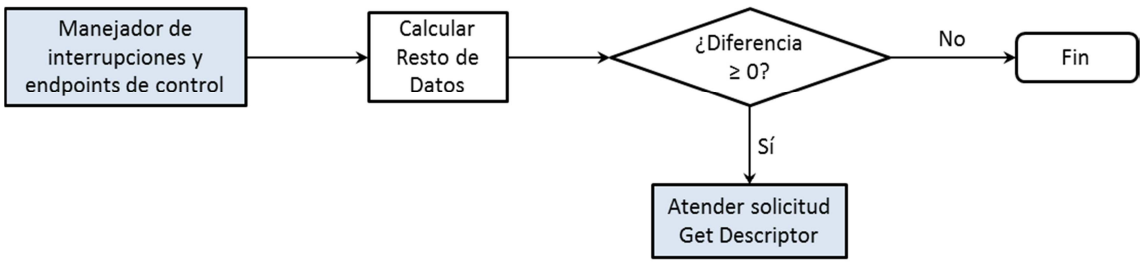


Figura 3.2.12 Diagrama de flujo de escribir más datos

Esta fase es la encargada de continuar con la respuesta a la solicitud en el *buffer* de entrada del *endpoint* seleccionado. Esto sucede cuando el tamaño de la respuesta es mayor que el tamaño máximo de paquete. La parte de la máquina de estados que describe esta etapa se muestra en la siguiente figura 3.2.13.

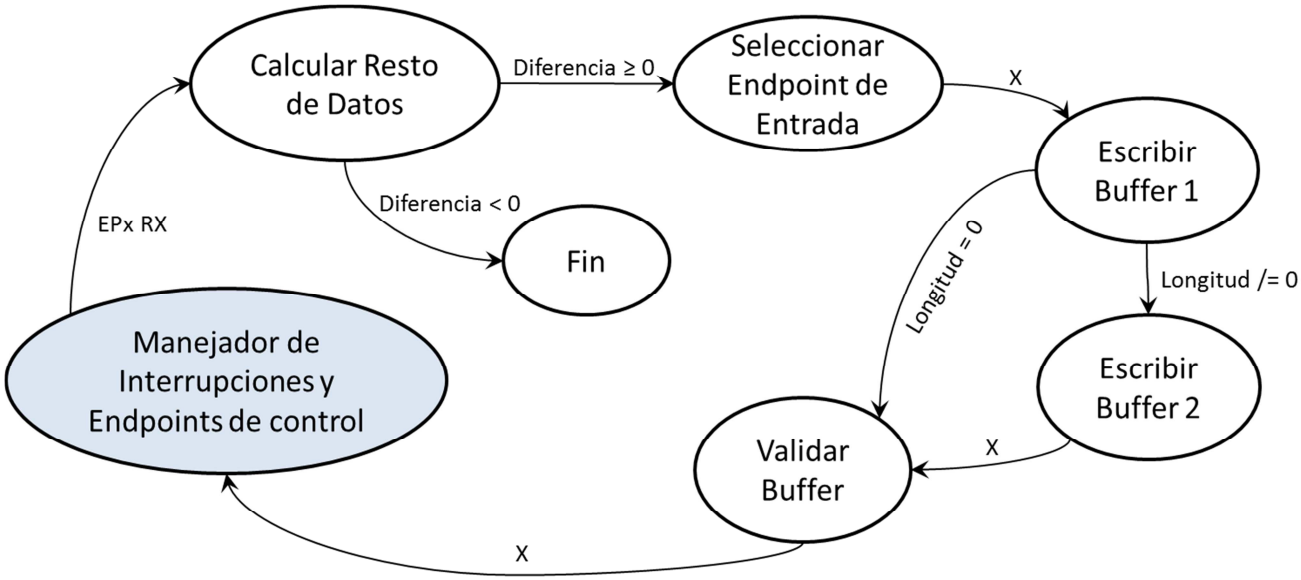


Figura 3.2.13 Escribir más datos de máquina de estados Principal

Como podemos ver en la imagen anterior, a esta etapa de la máquina de estados se llega desde la fase manejador de interrupciones y *endpoints* de control. También se vuelve a esta fase tras terminar de escribir los datos pendientes.

El primer estado de esta etapa es Calcular Resto de Datos, en el que se lleva a cabo el cálculo del número de datos de respuesta que quedan por enviar al *host* USB. Para ello se realiza la resta entre el valor de *wLength*, que es el número total de bytes de la respuesta a la solicitud, y el número de bytes enviados hasta este momento. Si la diferencia es menor que cero, el siguiente estado será Fin. Si la diferencia es cero se pone la variable longitud a 0, que indica que no quedan datos por enviar, y el siguiente estado será Seleccionar *Endpoint* de Entrada. Cuando la diferencia es mayor que cero, también se pasa al estado Seleccionar *Endpoint* de Entrada.

A partir de este momento, los estados por los que se pasa serán los mismos que para la etapa anterior *Get Descriptor*, ya que no deja de ser una prolongación de dicha etapa. La única diferencia es el número de bytes que se enviarán en este caso, que corresponde a la diferencia hallada anteriormente. Para la solicitud del descriptor de dispositivo, faltarían por enviar dos bytes, que corresponde a la diferencia entre el tamaño del descriptor de dispositivo (18 bytes) y el tamaño máximo de paquete (16 bytes).

3.3 Máquina de Estados Ejecutar Comando

La máquina de estados Ejecutar Comando es una máquina hija de la máquina de estados Principal. Esta se encarga de enviar los comandos que la máquina principal decide en cada momento, así como de enviar y recibir los datos que corresponden a cada uno de los comandos.

A continuación se procede a explicar la interfaz y descripción detallada de la máquina de estados Ejecutar Comando.

3.3.1 Interfaz entrada/salida

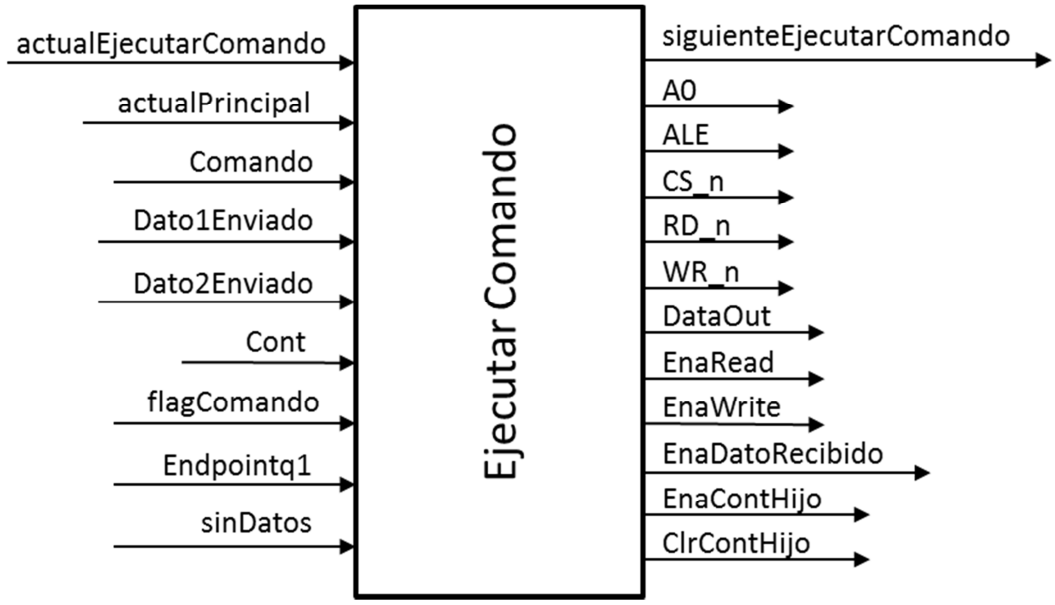


Figura 3.3.1 Interfaz de la máquina de estados Ejecutar Comando

Entradas:

- ActualEjecutarComando: Señal que indica el estado actual de la máquina de estados Ejecutar Comando.
- ActualPrincipal: Señal que indica el estado actual de la máquina de estados Principal.
- Comando: Señal de 8 bits que guarda el valor del comando que corresponda enviar.
- Dato1Enviado: Señal de 8 bits que guarda el primer dato a enviar tras un determinado comando.
- Dato2Enviado: Señal de 8 bits que guarda el segundo dato a enviar para ciertos comandos.
- Cont: Señal de tipo entero con rango de 0 a 5000000. Almacena el valor del contador de ciclos de reloj para realizar una determinada espera. El valor máximo es 5000000 que corresponde a una espera de 100 ms.
- FlagComando: *Flag* cuyo valor indica si a continuación de un determinado comando se deben recibir (0) o enviar datos (1).
- Endpointq1: Señal de 8 bits para registrar el valor del *endpoint* seleccionado.

- SinDatos: Señal que indica sí tras un comando se realizan transferencias de datos.

Salidas:

- SiguienteEjecutarComando: Señal que indica el siguiente estado de la máquina de estados Ejecutar Comando.
- A0: Señal que indica si la información transferida al dispositivo son comandos o datos.
- ALE: Señal que habilita la multiplexación de direcciones y datos. En este caso no se usa.
- CS_n: Señal que selecciona el dispositivo.
- RD_n: Señal que habilita al FT120 para leer el bus de datos.
- WR_n: Señal que habilita al FT120 para escribir en el bus de datos.
- DataOut: Señal de 8 bit que contiene los valores de los comandos o datos que serán enviados al dispositivo.
- EnaRead: Señal que habilita la lectura del bus DATA.
- EnaWrite: Señal que habilita la escritura del bus DATA.
- EnaDatoRecibido: Señal que permite cargar el valor de los datos de entrada leídos en Dato1Recibido.
- EnaContHijo: Señal que habilita a las máquinas hija a usar el contador que se usa en las diferentes esperas del sistema.
- ClrContHijo: Señal de las máquinas de estados hija que limpia el valor del contador de espera del sistema.

3.3.2 Descripción

En primer lugar se muestra el diagrama de flujo que corresponde a la máquina de estados Ejecutar Comando. En él se resume el proceso completo que se lleva a cabo en dicha máquina de estados, y que será explicada en detalle a continuación.

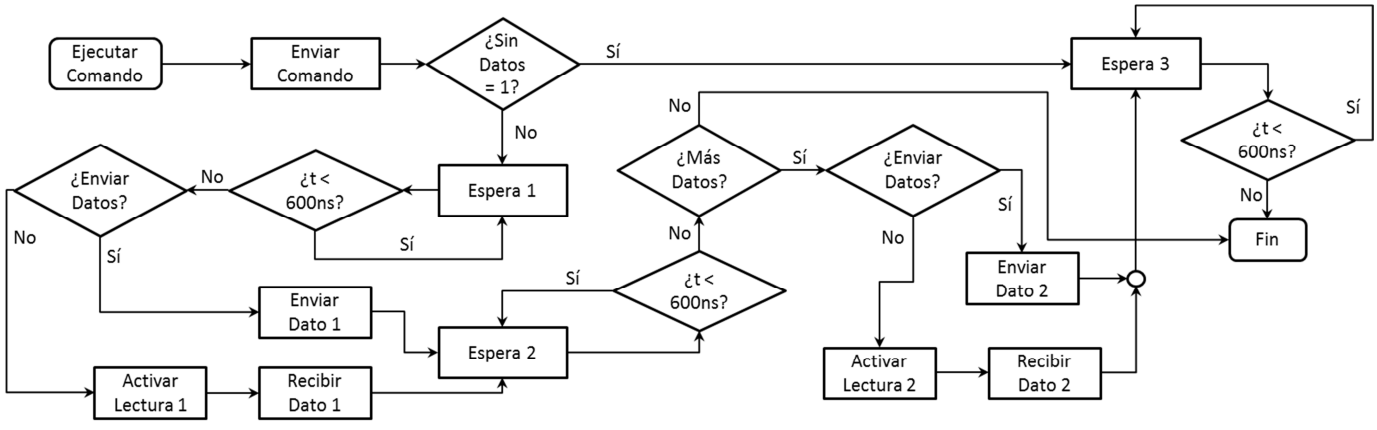


Figura 3.3.2 Diagrama de flujo de la máquina de estados Ejecutar Comando

Esta máquina de estados se ha dividido en tres partes: enviar comando, enviar datos y recibir datos. En el siguiente diagrama se detallan las conexiones entre las diferentes partes, estando cada una de ellas compuesta de varios estados.

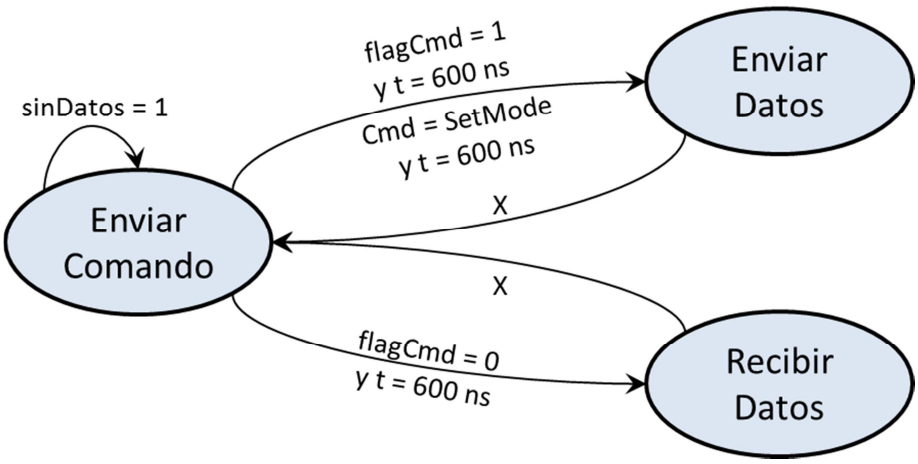


Figura 3.3.3 División en 3 partes de la máquina de estados Ejecutar Comando

A continuación se explican cada una de las partes anteriores y las conexiones que se establecen en ellas.

3.3.2.1 Enviar Comando

En esta primera etapa se realiza el envío de un byte que identifica uno de los comandos soportados por el FT120. A continuación se muestran los estados que componen esta parte.

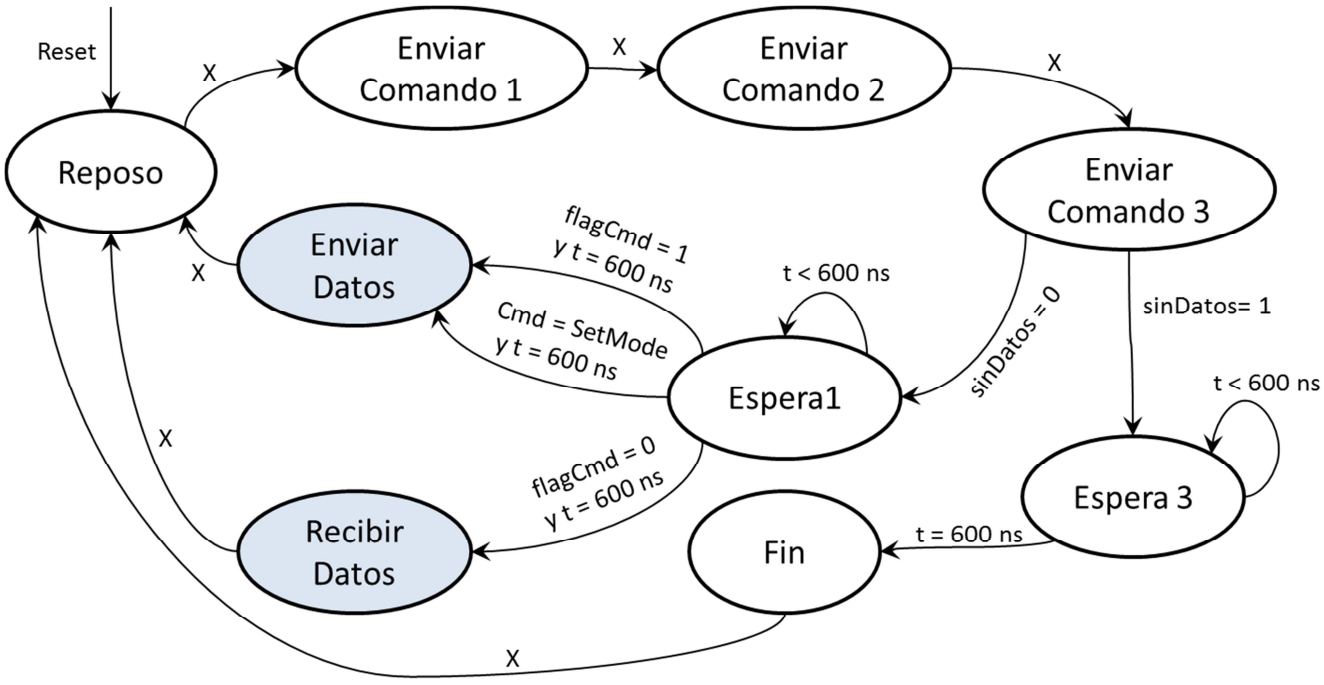


Figura 3.3.4 Enviar Comando de la máquina Ejecutar Comando

Esta máquina de estados parte del estado de Reposo y permanece así hasta que la máquina padre llega a alguno de los estados en los que se ha de enviar algún comando. El siguiente estado es siempre Enviar Comando 1.

En el estado Enviar Comando 1, se procede a activar las entradas del FT120 necesarias para escribir un comando en el bus de datos y que el dispositivo lo lea correctamente. Se activa el

pin *CS_n* para seleccionar el dispositivo, el pin *WR_n* para habilitar la escritura del bus, el pin *A0* se pone a 1 para indicar que lo que se escribe es un comando, se asigna el valor del comando a la señal intermedia *DataOut* y se activa *EnaWrite* para cargar el valor de *DataOut* en el bus de datos. Tras permanecer estos valores durante el ciclo de reloj correspondiente se pasa al siguiente estado Enviar Comando 2.

En el estado Enviar Comando 2, se realiza exactamente lo mismo que en el estado anterior. Por lo que sería equivalente a permanecer durante dos ciclos de reloj en el estado Enviar Comando 1. Esto se implementa así para que permanezcan los valores anteriores de los pines por lo menos 40 ns (2 ciclos). El siguiente estado es Enviar Comando 3.

En el estado Enviar Comando 3, se siguen manteniendo los valores anteriores excepto el pin *WR_n* que se desactiva en este estado. De esta manera, el comando permanecerá en el bus de datos un ciclo más desde la desactivación del *WR_n*, tal y como indica la especificación del FT120. En este estado se comprueba el *flag sinDatos*, que indica si tras el envío del comando debemos enviar/recibir datos o no. En el caso de que sí, su valor será 0, y el siguiente estado será Espera 1. Si por el contrario el *flag sinDatos* valiera 1, el siguiente estado sería Espera 3.

En el estado Espera 1 se realiza una espera de 600 ns, antes de proceder a enviar/recibir los datos correspondientes. Este tiempo es el indicado en la especificación y es implementado mediante un contador que cuenta hasta 30. Como el ciclo de reloj utilizado es de 20 ns, tras contar 30 ciclos de obtendrá la espera correspondiente de 600 ns. Tras dicha espera, se comprueba si el comando es un *SetMode* o el *flagComando* se encuentra activado. Si es así, se deberá realizar a continuación el envío de datos, y el siguiente estado será Enviar Dato 1.1 (en la fase Enviar Datos). En caso contrario, se deberá realizar la lectura de datos y el siguiente estado será Activar Lectura 1 (en la fase Recibir Datos).

En el estado Espera 3 se realiza una espera de 600 ns. Esta se implementa de igual manera que en el estado Espera 1. Cundo esta ha transcurrido, se pasa al siguiente estado Fin, que indicará que el envío del comando ha finalizado y que la máquina de estados Principal puede continuar.

3.3.2.2 Enviar Datos

En la etapa Enviar Datos, se realiza el envío de uno o dos datos para los comandos que lo requieran según la especificación del FT120. Los estados que detallan como se realiza dicho envío se muestran y explican a continuación.

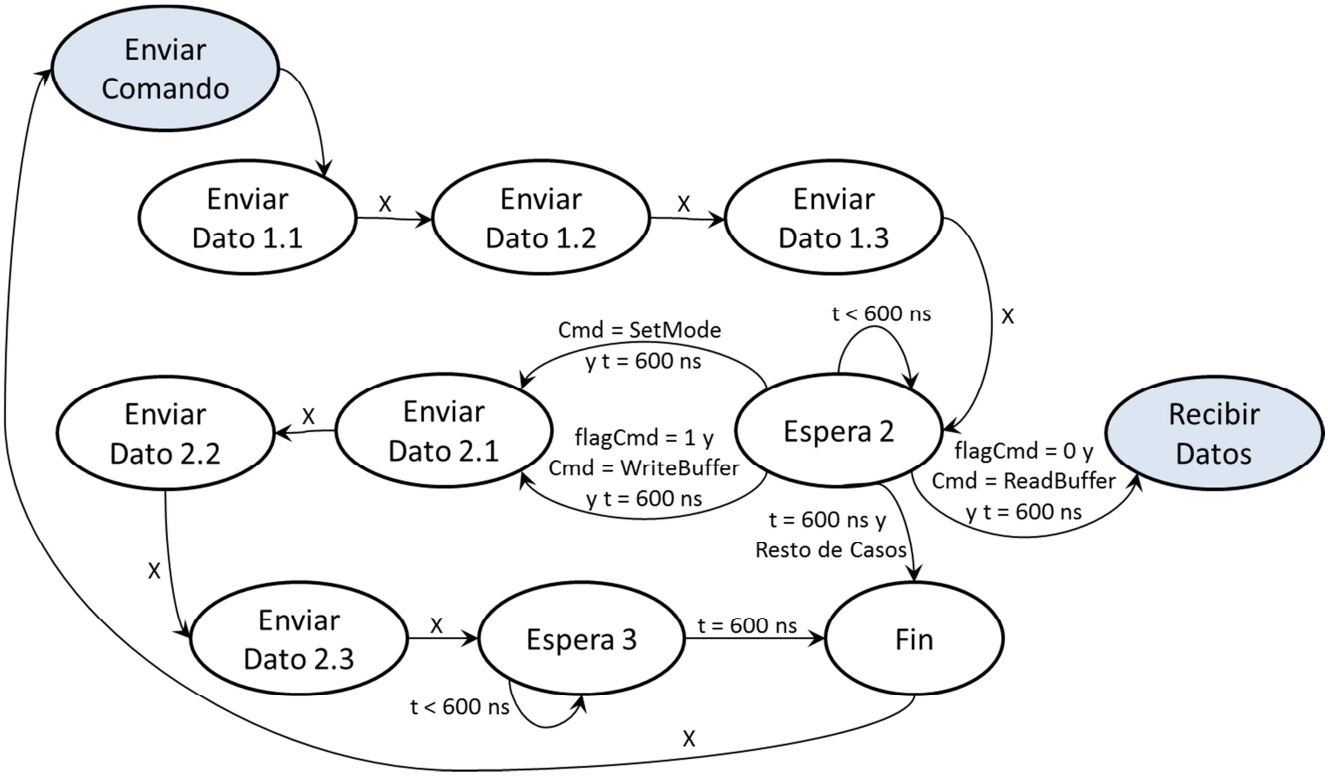


Figura 3.3.5 Enviar Datos de la máquina de estados Ejecutar Comando

El primer estado de esta fase es Enviar Dato 1.1, en donde se asigna a ciertas entradas del dispositivo el valor que permite escribir un dato en el bus de datos del dispositivo. Para ello se activa el pin *CS_n*, que selecciona el dispositivo, y el pin *WR_n*, que habilita la escritura del bus (igual que en el estado Enviar Comando 1.1). Se debe poner el pin *A0* a 0, para indicar que la información enviada corresponde a un byte de datos. También se le asigna el valor del dato a enviar a la señal intermedia *DataOut* y se activa el *flag EnaWrite* que permite cargar dicho valor en el bus de datos. Tras completarse todo esto, se pasa al siguiente estado Enviar Dato 1.2.

En el estado Enviar Dato 1.2, se realiza lo mismo que en el estado anterior, tal y como sucedía en la fase de Enviar Comando. El objetivo es el mismo, que los valores asignados permanezcan durante al menos dos ciclos de reloj. El siguiente estado será Enviar Dato 1.3.

El estado Enviar Dato 1.3 mantiene los valores asignados anteriormente con la única diferencia que se desactiva el pin *WR_n*. El motivo de esto es el mismo que en el estado Enviar Comando 3 de la fase Enviar Comando. Como se ha podido observar, el proceso para enviar un comando o un dato es el mismo, con la excepción del valor que se debe asignar al pin *A0*. El siguiente estado corresponde a Espera 2.

En el estado Espera 2 se vuelve a realizar una espera de 600 ns, tal y como se hacía en los estados Espera 1 y Espera 3. Después se comprueba que el dato corresponda al comando *SetMode*, o que corresponda a *WriteBuffer* y esté activado el flagComando. Si alguna de las condiciones se cumple se procederá al envío de un nuevo byte de datos, por lo que el siguiente estado será Enviar Dato 2.1. Cuando no se cumple lo anterior, se comprobará si se trata de un comando *ReadBuffer* y si se encuentra desactivado el flagComando, lo que implicará que el

siguiente estado será de la fase Recibir Datos. Para el resto de casos el siguiente estado será Fin, y se habrá terminado el envío de los datos del comando.

En los estados Enviar Comando 2.1, Enviar Comando 2.2 y Enviar Comando 2.3, se realiza el mismo procedimiento que para enviar el primer byte de datos del comando, con la diferencia que el dato que ahora se escribirá en el bus, será el que corresponda a este segundo envío. El paso entre estos estados se realiza de manera incondicional, y finalizarán en el estado Espera 3.

El estado Espera 3 es el mismo que en la fase Enviar Comandos, por lo que cuando transcurran 600 ns se pasará al estado de Fin.

3.3.2.3 Recibir Datos

En la etapa Recibir Datos, se recibe uno o dos datos proporcionados por el dispositivo tras enviar determinados comandos. Para ello se realizan las lecturas del bus de datos que correspondan. Los estados de esta etapa se muestran en la siguiente figura 3.3.6.

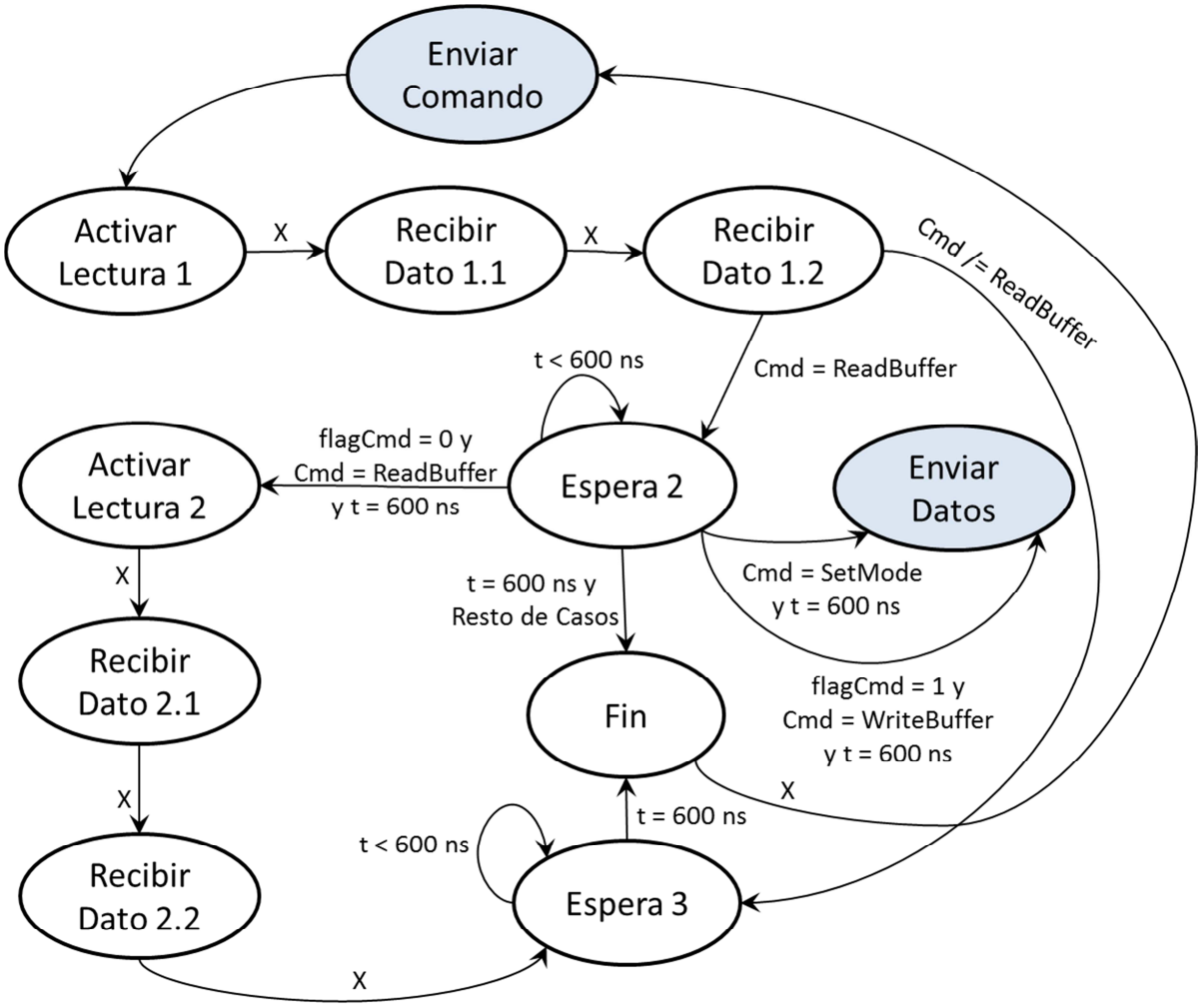


Figura 3.3.6 Recibir Datos de la máquina de estados Ejecutar Comando

En el estado Activar Lectura 1, se activan ciertas señales que inician el proceso de lectura, ya que informan al dispositivo de que se está preparado para recibir el byte de datos que

corresponda. Para ello se activa la señal *CS_n*, que selecciona el chip, se activa la señal *RD_n*, que habilita la lectura del bus, y *A0* se pone a 0 para indicar que son datos. En este estado no se activa el *flag EnaRead*, para no realizar la lectura todavía. El siguiente estado será Recibir Dato 1.1.

En el estado Recibir Dato 1.1 se realiza la lectura del dato que el dispositivo escribe en el bus de datos. Para ello se mantienen las señales a los valores del estado anterior, pero activando además la señal *EnaRead*. Esto permitirá que se realice la lectura del bus y se almacenará dicho valor. El siguiente estado es Recibir Dato 1.2.

El estado Recibir Dato 1.2 es igual al estado anterior, por lo que sería equivalente estar dos ciclos de reloj en dicho estado. Esto permite que los valores fijados anteriormente se mantengan durante dos ciclos de reloj, 40 ns. El motivo de esta implementación es no funcionar con valores límite, que serían los 20 ns de un ciclo que fija la especificación. En este estado se comprueba si el dato recibido pertenece al comando *ReadBuffer*, si es así el siguiente estado será Espera 2, en caso contrario será Espera 3.

El estado Espera 2 es el mismo que en la fase Enviar Datos. Cuando el comando correspondiente es *ReadBuffer* y se encuentra desactivado el flagComando, el siguiente estado será Activar Lectura 2. Para el resto de casos se actuará tal y como se indicaba anteriormente.

En los estados Activar lectura 2, Recibir Dato 2.1 y Recibir Dato 2.2, se lleva a cabo lo mismo que para el primer byte de datos recibido. La única diferencia será que el dato recibido será el correspondiente a la segunda escritura del dispositivo. Tras pasar por estos tres estados en el orden indicado, se pasará al estado Espera 3.

El estado Espera 3 es el mismo que en las dos fases anteriores, por lo que cuando transcurran los 600 ns correspondientes se pasará al estado de Fin.

3.4 Máquina de Estados Leer *Buffer*

Leer *Buffer* es una máquina de estados hija de la máquina Principal. Esta es la encargada de realizar las lecturas consecutivas necesarias en el estado Leer *Buffer* 2 de la máquina padre. En ella se pueden hacer tantas lecturas como el dispositivo indique, siempre que no se supere el tamaño máximo de paquete del *endpoint*.

A continuación se procede a explicar la interfaz y descripción detallada de la máquina de estados Leer *Buffer*.

3.4.1 Interfaz entrada/salida

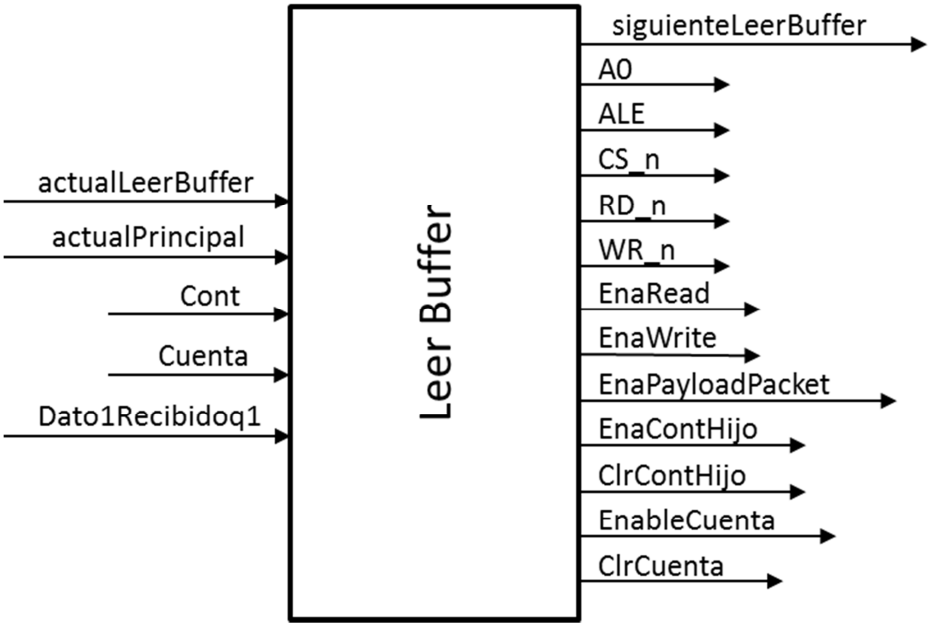


Figura 3.4.1 Interfaz de la máquina de estados Leer *Buffer*

Entradas:

- ActualLeerBuffer: Señal que indica el estado actual de la máquina de estados Leer *Buffer*.
- ActualPrincipal: Señal que indica el estado actual de la máquina de estados Principal.
- Cont: Señal de tipo entero con rango de 0 a 5000000. Almacena el valor del contador de ciclos de reloj para realizar una determinada espera. El valor máximo es 5000000 que corresponde a una espera de 100 ms.
- Cuenta: Señal de tipo entero que contiene el valor del contador que cuenta el número de paquetes recibidos/enviados cuando se lee/escribe en el *buffer*.
- Dato1Recibidoq1: Señal de 8 bits para registrar el valor leído del bus de datos.

Salidas:

- siguienteLeerBuffer: Señal que indica el siguiente estado de la máquina de estados Leer *Buffer*.

- A0: Señal que indica si la información transferida al dispositivo son comandos o datos.
- ALE: Señal que habilita la multiplexación de direcciones y datos. En este caso no se usa.
- CS_n: Señal que selecciona el dispositivo.
- RD_n: Señal que habilita al FT120 para leer el bus de datos.
- WR_n: Señal que habilita al FT120 para escribir en el bus de datos.
- EnaRead: Señal que habilita la lectura del bus DATA.
- EnaWrite: Señal que habilita la escritura del bus DATA.
- EnaPayloadPacket: Señal que permite cargar el valor de los paquetes de entrada en PayloadPacketR durante la lectura del *buffer*.
- EnaContHijo: Señal que habilita a las máquinas hija a usar el contador que se usa en las diferentes esperas del sistema.
- ClrContHijo: Señal de las máquinas de estados hija que limpia el valor del contador de espera del sistema.
- EnableCuenta: Señal que habilita el contador que realiza la cuenta de los paquetes recibidos/enviados cuando se lee/escribe en el *buffer*.
- ClrCuenta: Señal que limpia el valor de la cuenta del contador de paquetes recibidos/enviados.

3.4.2 Descripción

Primeramente se muestra el diagrama de flujo que corresponde a la máquina de estados Leer *Buffer*, en el cual se puede observar el proceso que se lleva a cabo para realizar una lectura del *buffer*.

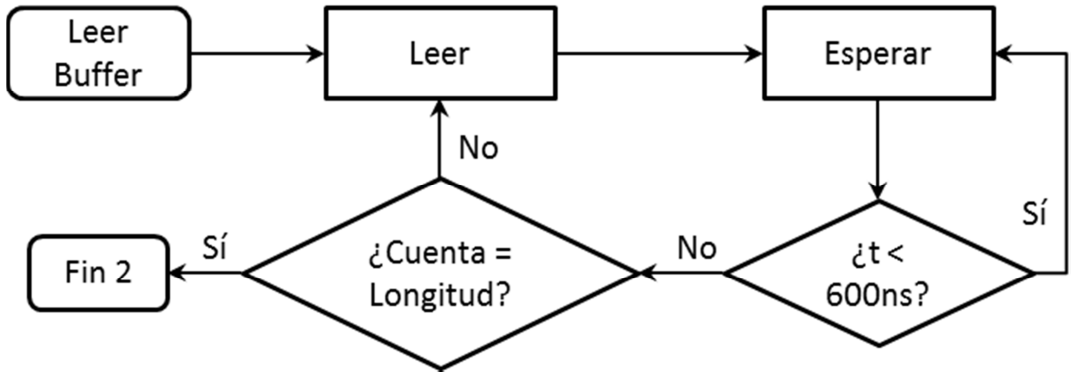


Figura 3.4.2 Diagrama de flujo de la máquina de estados Leer *Buffer*

A continuación, también se representa el diagrama de la máquina de estados implementada, y una explicación detallada del comportamiento del sistema en cada uno de los estados de los que se compone.

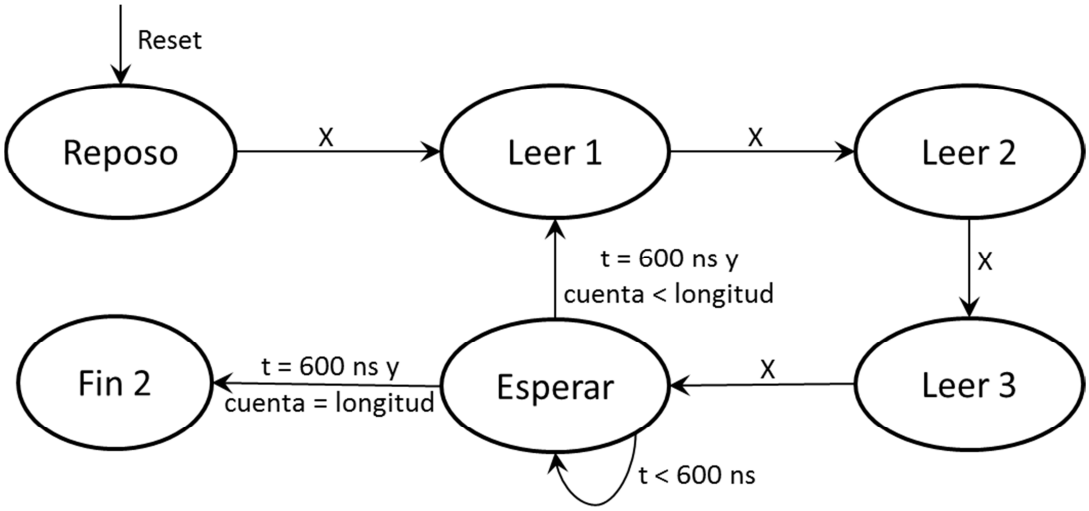


Figura 3.4.3 Máquina de estados Leer *Buffer*

La máquina de estados Leer *Buffer* parte del estado Reposo, del cual se saldrá cuando la máquina de estados Principal llegue al estado Leer *Buffer* 2. En este estado se realiza un borrado (*clear)* del contador de datos recibidos. El siguiente estado es siempre Leer 1.

En el estado Leer 1, se activan las entradas del dispositivo necesarias para preparar la lectura del *buffer* de datos. La señal *CS_n* se activa para seleccionar el chip y la señal *RD_n* para habilitar la lectura del bus de datos. Se indica que lo leído es un dato con *A0* puesto a 0. En este estado aún no se realiza la lectura, se da tiempo a que el dispositivo escriba en el bus. El siguiente estado es Leer 2.

En el estado Leer 2, se mantienen los valores de los pines asignados en el estado anterior. Además se activa la señal *EnaRead*, que permitirá leer el dato escrito por el dispositivo en el bus de datos. El estado siguiente será Leer 3.

En el estado Leer 3 se mantiene todo igual que en el estado anterior. Esto permite que los valores asignados permanezcan durante dos ciclos de reloj completos. El siguiente estado es Esperar.

El estado Esperar se utiliza para realizar una espera de 600 ns entre las distintas lecturas. Para ello se utiliza un contador de 30 ciclos de reloj. Cuando ha transcurrido este tiempo, se comprueba si faltan lecturas por realizar, para lo que se usa un nuevo contador. El número de lecturas a realizar viene fijado por la variable longitud, que adquiere su valor en la máquina de estados Principal. Se comprueba si el número de lecturas es igual al valor de longitud. Cuando esto se cumpla, indicará que hemos terminado de recibir los datos enviados por el FT120, y el siguiente estado será Fin 2. Si por el contrario no se cumple, se deberá realizar una nueva lectura e incrementar el contador de lecturas realizadas. En este caso el siguiente estado será nuevamente Leer 1.

En el estado Fin 2 se realiza un nuevo *clear* del contador, y permite que la máquina de estados Principal continúe con su ejecución.

3.5 Máquina de Estados Escribir *Buffer*

La máquina de estados Escribir *Buffer* es una hija de la máquina Principal. En ella se realizan las escrituras necesarias para enviar la información solicitada por el dispositivo, que corresponden al estado Leer *Buffer* 2 de la máquina padre. Como se puede observar en la siguiente figura 3.5.3, la máquina es similar a la anterior Leer *Buffer*, cambiando los estados de lectura por los de escritura.

A continuación se procede a explicar la interfaz y descripción detallada de la máquina de estados Escribir *Buffer*.

3.5.1 Interfaz entrada/salida

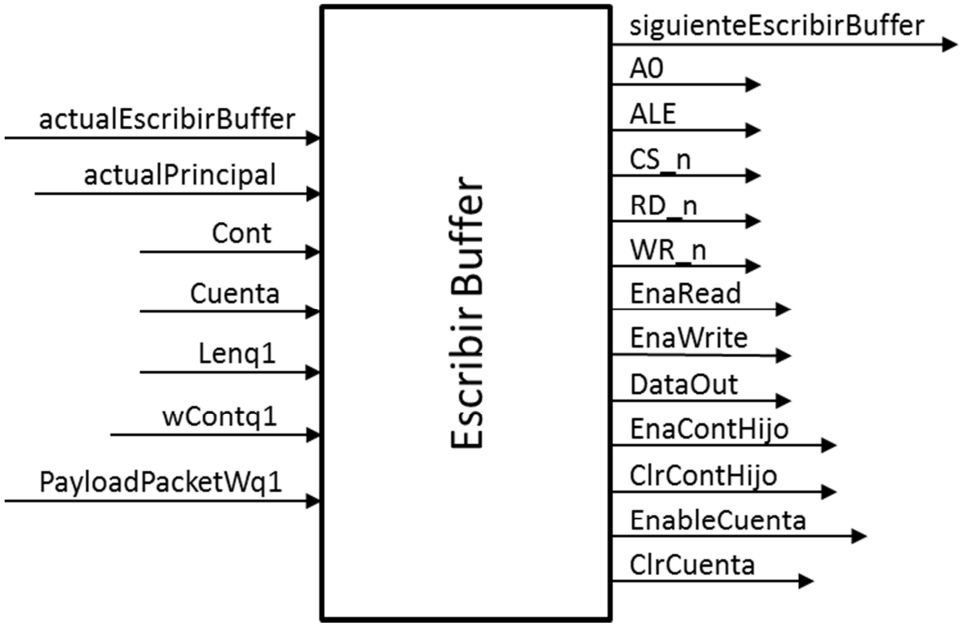


Figura 3.5.1 Interfaz de máquina de estados Escribir *Buffer*

Entradas:

- ActualEscribirBuffer: Señal que indica el estado actual de la máquina de estados Escribir *Buffer*.
- ActualPrincipal: Señal que indica el estado actual de la máquina de estados Principal.
- Cont: Señal de tipo entero con rango de 0 a 5000000. Almacena el valor del contador de ciclos de reloj para realizar una determinada espera. El valor máximo es 5000000 que corresponde a una espera de 100 ms.
- Cuenta: Señal de tipo entero que contiene el valor del contador que cuenta el número de paquetes recibidos/enviados cuando se lee/escribe en el *buffer*.
- Lenq1: Señal de 8 bits para registrar el número de paquetes que faltan por enviar en la escritura del *buffer*.
- wCountq1: Señal de tipo entero para registrar el número de paquetes que se han enviado hasta el momento del total que se deben enviar en la escritura del *buffer*.

- PayloadPacketWq1: Señal de 8 bits para registrar la carga de los paquetes escritos, en la escritura del *buffer*.

Salidas:

- SiguienteEscribirBuffer: Señal que indica el siguiente estado de la máquina de estados Escribir *Buffer*.
- A0: Señal que indica si la información transferida al dispositivo son comandos o datos.
- ALE: Señal que habilita la multiplexación de direcciones y datos. En este caso no se usa.
- CS_n: Señal que selecciona el dispositivo.
- RD_n: Señal que habilita al FT120 para leer el bus de datos.
- WR_n: Señal que habilita al FT120 para escribir en el bus de datos.
- EnaRead: Señal que habilita la lectura del bus DATA.
- EnaWrite: Señal que habilita la escritura del bus DATA.
- DataOut: Señal de 8 bit que contiene los valores de los comandos o datos que serán enviados al dispositivo.
- EnaContHijo: Señal que habilita a las máquinas hija a usar el contador que se usa en las diferentes esperas del sistema.
- ClrContHijo: Señal de las máquinas de estados hija que limpia el valor del contador de espera del sistema.
- EnableCuenta: Señal que habilita el contador que realiza la cuenta de los paquetes recibidos/enviados cuando se lee/escribe en el *buffer*.
- ClrCuenta: Señal que limpia el valor de la cuenta del contador de paquetes recibidos/enviados.

3.5.2 Descripción

En la siguiente figura se muestra el diagrama de flujo correspondiente a la máquina de estados Escribir *Buffer*, en el que se muestra el proceso para llevar a cabo una escritura del *buffer*.

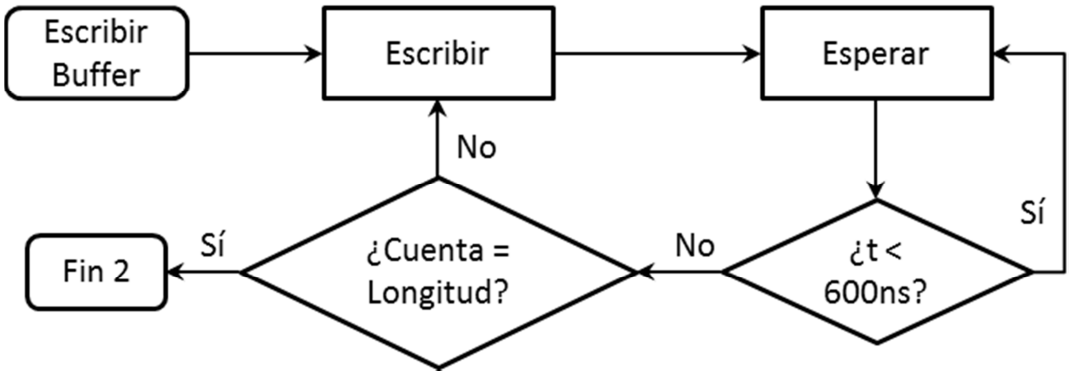


Figura 3.5.2 Diagrama de flujo de la máquina de estados Escribir *Buffer*

A continuación se incluye también el diagrama de la máquina de estados, y se realiza una explicación detallada del comportamiento del sistema en cada uno de los estados.

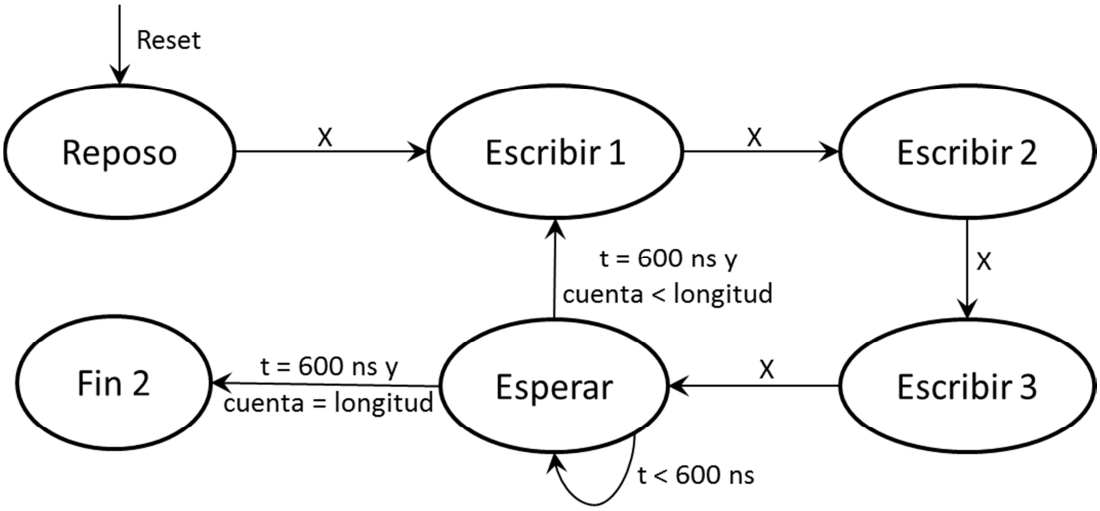


Figura 3.5.3 Máquina de estados Escribir *Buffer*

El estado inicial es el Reposo, del cual se sale cuando la máquina de estados padre llega al estado Escribir *Buffer* 2. Se realiza un *clear* del contador de datos enviados y se pasa al siguiente estado Escribir 1.

En el estado Escribir 1, se da el valor a las entradas del dispositivo que permiten realizar una escritura de un dato en el bus de datos. Se activa el pin *CS_n* para seleccionar el dispositivo, se activa *WR_n* para habilitar la escritura y se pone *A0* a 0 para indicar que lo que se envía es un dato. Además se asigna a la señal *DataOut* el byte de escritura que corresponda en dicho momento, y se habilita el *flag EnaWrite* que cargará en el bus de datos el valor de dicha señal. Una vez realizada la escritura se pasa al estado Escribir 2.

El Estado Escribir 2 es igual al estado anterior, Escribir 1. Esto se repite para que los valores asignados anteriormente permanezcan durante dos ciclos de reloj. El siguiente estado es Escribir 3.

En el estado Escribir 3 se deshabilita la entrada del dispositivo *WR_n*, indicando que termina la escritura del dato. El resto de valores deben permanecer durante este ciclo extra, para cumplir con lo indicado en la especificación. En este estado se incrementa el contador de escrituras realizadas, que comprobaremos en el siguiente estado Esperar.

En el estado Esperar, se usa un contador para realizar una espera de 600 ns que separe las diferentes escrituras realizadas en el *buffer*. Se debe realizar además una comprobación del número de escrituras consecutivas realizadas hasta el momento, para lo que se usa otro contador. El número de escrituras viene fijado por la variable longitud, que se le da valor en la máquina de estado principal. Cuando el contador de escrituras alcance el valor de longitud, se habrán realizado todos los envíos necesarios y se pasará al estado Fin 2. Si el número de escrituras es inferior a longitud, se continuarán realizando los envíos necesarios y el siguiente estado volverá a ser Escribir 1.

El estado Fin 2 es igual al caso anterior, se realiza un *clear* del contador y la máquina Principal continuará con su ejecución.

3.6 Otros Componentes

En este apartado se detallan el resto de componentes que componen el sistema implementado en este proyecto.

3.6.1 Contadores

3.6.1.1 Interfaz entrada/salida

La interfaz que corresponde a ambos contadores utilizados en este proyecto se muestra y detalla a continuación.

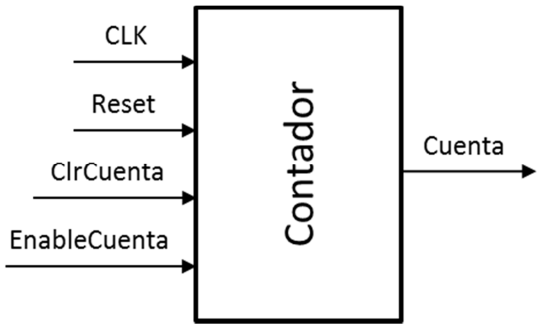


Figura 3.6.1 Interfaz de los contadores

Entradas:

- CLK: Señal de reloj.
- Reset: Señal de reinicio del contador.
- ClrCuenta: Señal de puesta a 0 del valor del contador.
- EnableCuenta: Señal que habilita la cuenta del contador.

Salidas:

- Cuenta: Señal que contiene el valor del contador.

3.6.1.2 Descripción

El contador incrementa la cuenta cuando recibe un flanco de la señal de reloj, el Reset se encuentra desactivado (Reset=1), el ClrCuenta no ha sido activado (ClrCuenta=0) y el EnableCuenta si se ha activado (EnableCuenta =1). Su funcionamiento se basa en un sumador que suma de uno en uno.

Dos de estos bloques son utilizados en este proyecto. El Contador 1, que realiza la cuenta de los ciclos de reloj correspondientes a una espera determinada. El Contador 2, que cuenta el número de lecturas o escrituras que componen la lectura o escritura del *buffer*.

3.6.2 Detector de Flanco

3.6.2.1 Interfaz entrada/salida

La interfaz que corresponde al componente utilizado Detector de Flanco se muestra y detalla a continuación.

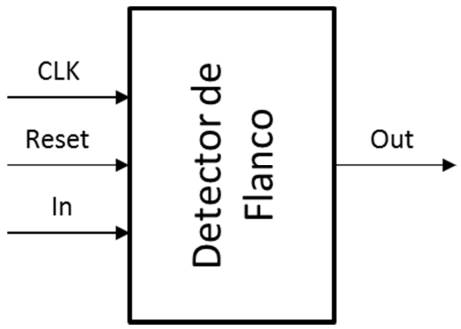


Figura 3.6.2 Interfaz del Detector de Flanco

Entradas:

- CLK: Señal de reloj.
- Reset: Señal de reinicio del Detector de Flanco.
- In: Señal que se quiere detectar.

Salidas:

- Out: Señal detectada.

3.6.2.2 Descripción

El bloque Detector de Flanco se encarga de generar una señal de duración un ciclo de reloj cuando se detecta que la señal de entrada ha sido activada. Este bloque se utiliza en las señales de entrada introducidas a partir de los botones de la FPGA.

3.6.3 Registros

3.6.3.1 Interfaz entrada/salida

La interfaz que corresponde al bloque Registros se muestra y detalla a continuación.

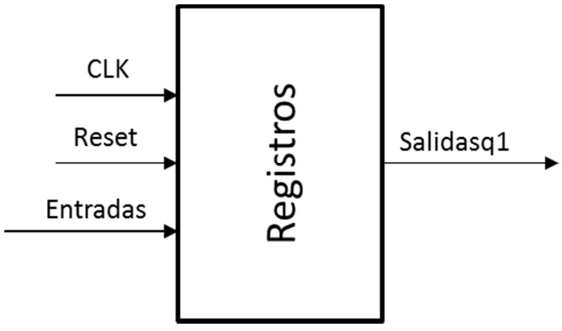


Figura 3.6.3 Interfaz de los Registros

Entradas:

- CLK: Señal de reloj.
- Reset: Señal de reinicio del bloque Registros.
- Entradas: Señales cuyo valor se desea cargar a la salida de este bloque.

Salidas:

- Salidasq1: Señales que contienen los valores registrados de entrada.

3.6.3.2 Descripción

El bloque Registros se encarga de asignar a las señales Salidasq1 el valor que contienen las correspondientes señales de entrada, cuando se recibe un flanco de reloj y la señal de *reset* se encuentra desactivada.

Este bloque permite la evolución de las máquinas de estados, ya que se encarga de asignar al siguiente estado el valor del estado actual. Además también permite que se mantenga el valor de otras muchas señales al registrarlas mediante la utilización de biestables realimentados. En definitiva, permite el almacenamiento de información durante la ejecución de este proyecto.

4. Validación

4.1 Simulación

En la ejecución de este proyecto se ha usado el programa ModelSim, que es un simulador de circuitos digitales descritos en lenguajes HDL. En este caso, se ha utilizado para compilar y simular el código VHDL diseñado para la realización de este proyecto.

El primer paso para poder simular el funcionamiento del sistema implementado, es la creación de un banco de pruebas. Este debe simular el comportamiento del dispositivo FT120, con el cual se comunica el sistema grabado en la FPGA. En función de los valores de las entradas del dispositivo, el banco de pruebas escribe en el bus de datos la información que corresponda.

Una vez implementado el banco de pruebas, se procede a realizar la simulación y a comprobar si el funcionamiento es el correcto. A continuación, indicamos algunas de las simulaciones más importantes:

4.1.1 Enviar Comando

El sistema implementado debe enviar diferentes comandos al FT120, cuyo procedimiento es el mismo en todos los casos. A continuación se muestra una captura de la simulación del envío de un comando.

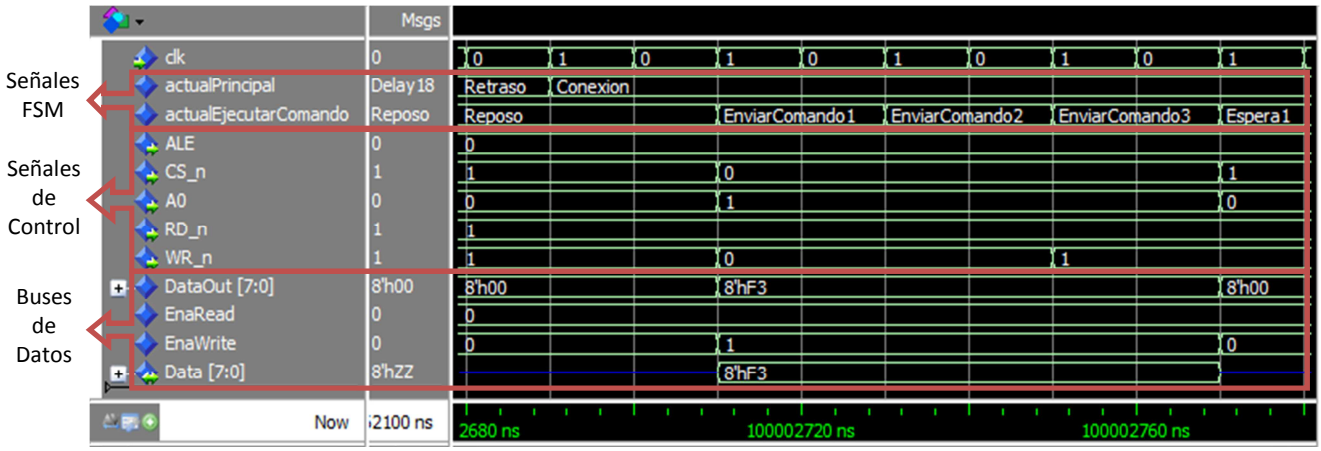


Figura 4.1.1 Simulación enviar comando

En este ejemplo se muestra el envío del comando *Set Mode*, que corresponde al código F3h en la señal *DataOut* y que se envía durante el estado conexión. Para realizar el envío se pone el pin *CS_n* a 0, que selecciona el dispositivo, *A0* a 1, que indica que lo enviado es un comando, y *WR_n* a 0, que habilita la escritura del bus de datos. La señal *DataOut* contiene el valor del comando a enviar en este estado, y se carga en el bus *Data* cuando se activa la señal *EnaWrite*. Estos valores se mantienen durante dos ciclos consecutivos (40ns), ya que la especificación indica que el tiempo mínimo que debe estar el dato es 30 ns. Durante el tercer ciclo permanecen todos los valores iguales a excepción de *WR_n*, que es desactivado. El valor del

comando permanece en el bus durante este ciclo, cumpliendo así los 10 ns mínimos que indica la especificación.

Una vez realizada esta simulación, se comprueba que los valores de la señales corresponden con lo esperado, por lo que se pude asumir que a priori se ha diseñado correctamente el envío de un comando.

4.1.2 Enviar Dato

Tras el envío de algunos comandos, el sistema debe proceder a enviar uno o dos datos con la información que corresponda. A continuación se muestra la simulación del envío de uno de los datos.

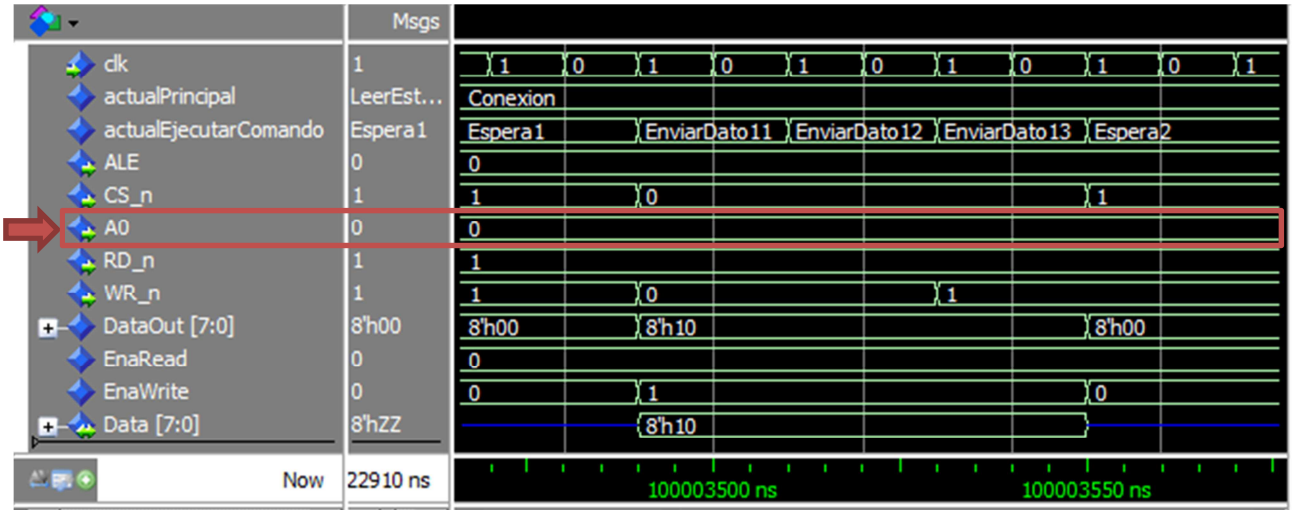


Figura 4.1.2 Simulación enviar dato

En este ejemplo continuamos en el estado anterior de conexión y el dato enviado es 10h, que asigna la configuración necesaria. El procedimiento para enviar datos al dispositivo FT120 es el mismo que para enviar comandos, con la excepción de que *A0* debe valer 0, que indica que la información enviada corresponde a un dato.

Al igual que en el aparatado anterior, se comprueba que los resultados obtenidos son los correctos, y que la simulación del envío de un dato se corresponde con la especificación.

4.1.3 Leer Dato

El sistema lee los datos de respuesta del dispositivo tras el envío de algunos comandos. Para comprobar cómo se realiza la lectura, se muestra a continuación una simulación.

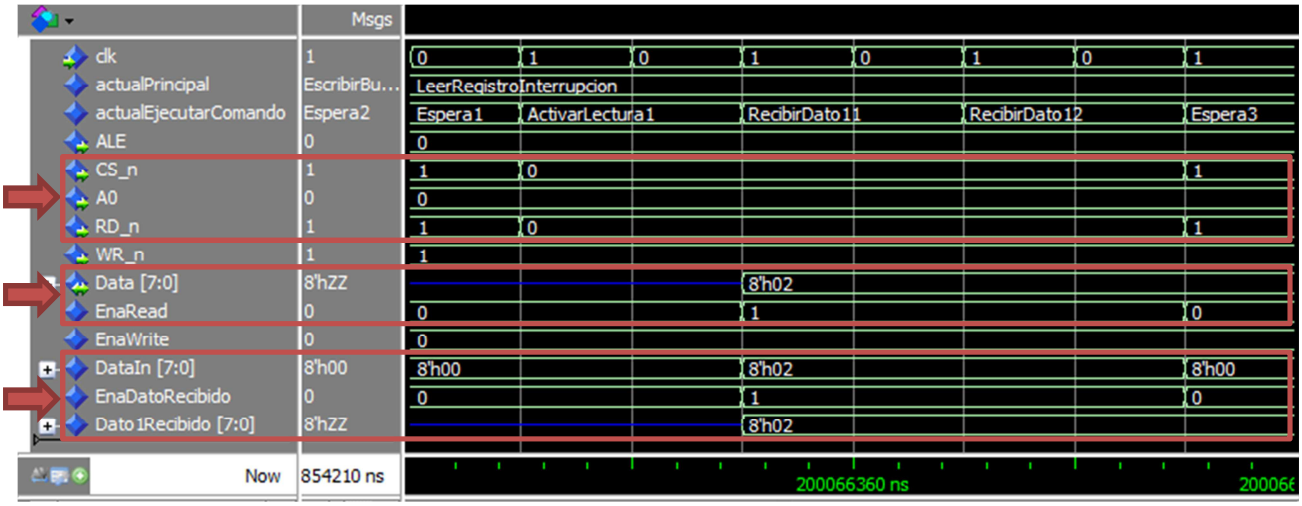


Figura 4.1.3 Simulación leer dato

En la figura anterior se muestra la lectura de un dato tras el envío del comando *Read Interrupt Register* (F4h) en el estado *Leer Registro de Interrupción*. El dato leído es 02h, que indica que la interrupción recibida es para el *endpoint 0* de entrada. Para realizar la lectura se pone *CS_n* a 0, que selecciona el dispositivo, *A0* a 0, que indica que lo leído es un dato, y *RD_n* a 0, que habilita la lectura del bus de datos. Tras un ciclo de reloj como máximo, el dispositivo carga el dato en el bus *Data*, cuyo valor se guarda en la señal *DataIn* cuando se activa la señal *EnaRead*. El valor obtenido será registrado y almacenado, para ello se activa la señal *EnaDatoRecibido* y se guarda en el registro *Data1Recibido*. Estos valores se mantienen durante dos ciclos de reloj para cumplir con lo indicado en la especificación.

Se ha comprobado que al simular la lectura de un dato, los resultados obtenidos concuerdan con lo que se pretendía diseñar, y no se ha detectado ningún problema.

4.1.4 Leer/Escribir *buffer*

Otras de las simulaciones que se han realizado son la lectura y escritura del *buffer*. Para ello se realizan múltiples lecturas o escrituras consecutivas, que se realizan siguiendo el mismo procedimiento que se ha explicado anteriormente. A continuación se muestra la simulación de la lectura del *buffer*, compuesta de 8 lecturas, que corresponde a la longitud de carga útil del paquete indicada por el dispositivo.

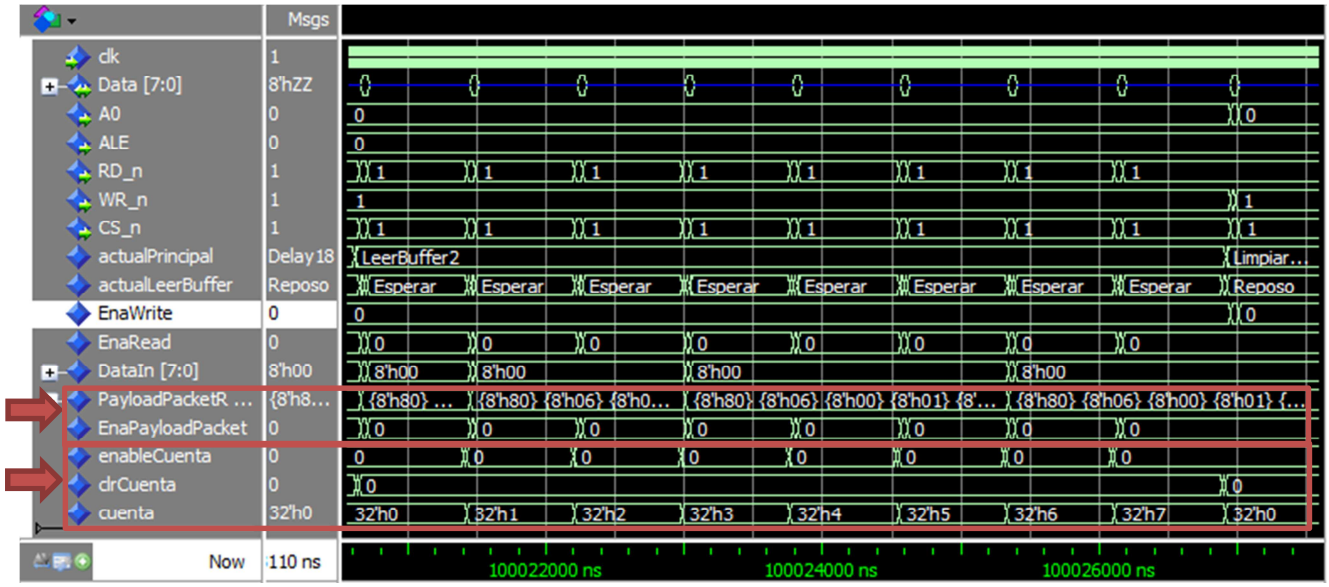


Figura 4.1.4 Simulación leer *buffer*

Cada una de las lecturas se realiza del mismo modo que en el apartado anterior *Leer Dato*, con la diferencia de que los datos se guardan en la señal *PayloadPacketR* al activar *EnaPayloadPacket*. En la figura anterior podemos comprobar que se realizan 8 lecturas consecutivas, que se contabilizan mediante el uso de un contador. La señal *cuenta* es la encargada de almacenar el valor del contador, que se pone a cero cuando se activa la señal *clrCuenta*, y se incrementa al activar *EnableCuenta*. Estas señales se muestran en la simulación y se comprueba que funcionan correctamente.

En el caso de *Escribir Buffer*, el procedimiento es el mismo que para *Leer Buffer*, con la única diferencia de que lo que se realizan son escrituras de forma consecutiva.

Con esta simulación comprobamos que el sistema implementado realiza la lectura del *buffer* tal y como se ha indicado en el apartado de diseño.

4.1.5 Espera entre lecturas o escrituras.

Según indica la especificación del FT120, debe pasar un tiempo de al menos 600 ns entre las lecturas o escrituras en el bus de datos. Por este motivo cada vez que se realiza una lectura o una escritura, ya sea de comandos o de datos, se realiza dicha espera. Para implementarlas se utiliza un contador que cuenta tantos ciclos de reloj como correspondan. En el caso de 600ns se contarán 30 ciclos, al ser 20ns el periodo de reloj. En la siguiente figura se muestra una de las esperas realizadas.

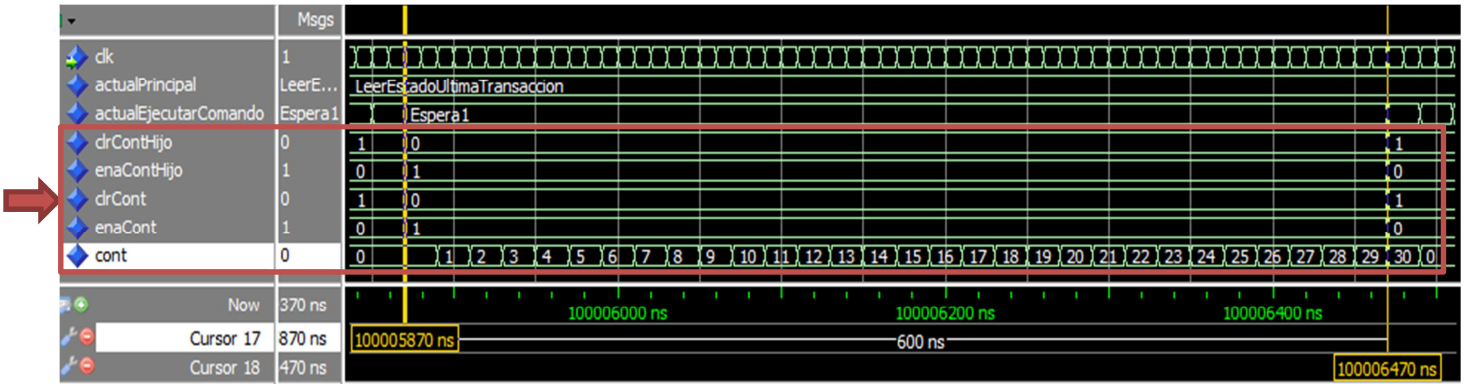


Figura 4.1.5 Simulación espera entre lecturas y escrituras

La señal *cont* es la que contiene el valor del contador, que se incrementa cuando se activa *enaCont*, y se pone a 0 cuando se activa *clrCont*. En función de la máquina de estados que utilice el contador, *enaCont* y *clrCont* valdrán lo que indique *enaContHijo* y *clrContHijo*, o *enaContPadre* y *clrContPadre*. En este ejemplo, vemos que la espera se realiza en la máquina de estados hija *EjecutarComando*, por lo que las señales de control del contador corresponden a las indicadas por la máquina hija. En esta simulación podemos comprobar que se realiza correctamente la espera de los 30 ciclos de reloj, observando la evolución de la señal *cont*.

Al realizar la simulación comprobamos que la espera entre lecturas y escrituras se corresponde con lo diseñado, y que coincide con la especificación.

4.1.6 Evolución Máquinas de estados

Comprobar que el orden de los estados por los que pasa el programa concuerda con lo indicado en las máquinas de estados diseñadas. Si esto es así, significará que la comunicación entre la FPGA y el dispositivo concuerda con lo esperado según la especificación del USB y del FT120.

El resultado obtenido al realizar esta simulación, es que la máquina de estados se comporta tal y como se ha indicado en el anterior apartado de diseño.

4.1.7 Simulación global

Además de las simulaciones parciales indicadas anteriormente, se ha realizado una simulación del sistema completo. De esta manera se comprueba que el circuito diseñado funciona correctamente.

El banco de pruebas contiene inicialmente la desactivación de la señal *Reset* y la activación de la señal *start* que inician el sistema implementado. Cuando esto sucede la máquina de estados Principal saldrá del estado de reposo y comenzará la comunicación.

A continuación, el banco de pruebas realiza una espera de 100 ms, dando tiempo al circuito implementado a que realice la inicialización y configuración inicial. Posteriormente se activará la señal de interrupción *Int_n,* que permitirá que la máquina de estados Principal salga del

estado Espera Interrupción y continúe realizando las transferencias de los comandos correspondientes.

A partir de este momento, el banco de pruebas realizará continuamente comprobaciones de los valores de las señales de control *CS_n*, *A0*, *WR_n*, *RD_n* y *ALE.* Cuando estas correspondan al envío de un comando, lo almacenará en la variable *comando_aux*.

El siguiente paso será seguir comprobando los valores de las señales de control. Cuando estas indiquen la lectura de un dato, se comprueba el valor del último comando recibido y se envía el dato que corresponda para que el sistema realice su lectura. Para ello el banco de pruebas asigna al bus *Data* el dato que corresponde enviar según el comando recibido.

En el caso de que el comando recibido sea *Read Buffer,* el banco de pruebas deberá realizar el envío del paquete *setup*. Esto supone realizar el envío de varios datos hasta completar el envío de toda la información. Para ello se utiliza una variable entera *cnt*, que indicará el siguiente byte de datos del paquete *setup* que corresponde enviar. El procedimiento para realizar cada uno de los envíos será el mismo que para el resto de comandos recibidos.

4.2 Pruebas Hardware

Lo primero antes de realizar las pruebas sobre el sistema, es sintetizar el código VHDL, implementar el diseño correspondiente y grabarlo en la FPGA. Para ello se utiliza el programa Xilinx ISE, que permite llevar a cabo todo esto.

Una vez que el sistema se encuentra implementado a nivel hardware en la FPGA, se comprobará si el funcionamiento es el correcto y concuerda con lo simulado en el apartado anterior. Para ello se han utilizado algunos mecanismos de comprobación, que permiten verificar el comportamiento del sistema implementado. A continuación se detallan las pruebas realizadas y los mecanismos utilizados en cada una de ellas.

4.2.1 Mecanismos de comprobación

4.2.1.1 Leds y *switches*

Se usan los Leds y *switches* de la FPGA para mostrar los valores de ciertas señales y saber si el funcionamiento es el correcto. Para ello, se asigna el valor de cada bit del dato a mostrar a cada uno de los 8 leds disponibles. Mediante los *switches*, se seleccionan las diferentes señales que serán asignadas a los leds.

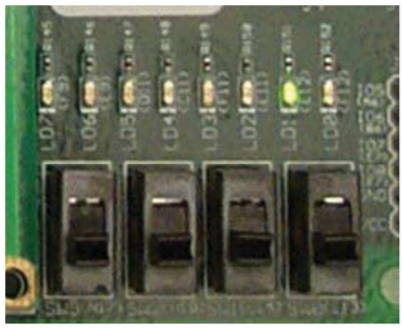


Figura 4.2.1 Leds y *switches*

La señal que principalmente mostramos por los leds es la señal depurar. Esta se diseña con el objetivo de indicar el estado en el que se encuentra la máquina de estados Principal en un determinado momento. Se debe tener en cuenta que la evolución del sistema es muy rápida, por lo que no es apreciable al ojo humano los valores por los que van pasando los leds. Aun así el valor del último estado nos proporciona mucha información, ya que tal y como está diseñado el sistema, el llegar a un determinado estado implica haber implementado los anteriores de manera correcta. Para hacer comprobaciones de los estados intermedios bastará con cambiar el diseño y finalizar el programa en dichos estados.

Otras de las señales que se han considerado importantes mostrar para comprobar el funcionamiento son: *Dato1Recibidoq1*, que muestra la última lectura realizada; *comando_auxq1*, que contiene el valor del siguiente comando; *flag1q1*, que indica que se ha completado la primera escritura del *buffer*; *Lenq1*, que contiene el número de paquetes que faltan por enviar; *wCountq1*, que contiene cuantos paquetes se han enviado hasta el momento; *wLengthq1*, que indica el total de paquetes a enviar; y cada uno de los bytes que componen *PayloadPacketRq1*, que contienen los datos obtenidos en la lectura del *buffer*. Se debe tener en cuenta que estas señales pueden ser modificadas para mostrar aquellas que se considere más oportuno en cada momento.

Para poder seleccionar las distintas señales enumeradas anteriormente, se usan los *switches* que proporciona la FPGA. Al disponer de 4 *switches*, permite la selección de 15 señales de

8bits, que nos proporciona bastante información respecto al estado de la ejecución del sistema.

4.2.1.2 USBlyzer

El USBlyzer es un analizador del protocolo USB, que permite capturar las tramas intercambiadas entre el *host* USB y el dispositivo conectado, en este caso el FT120.

El motivo por el que se ha utilizado, es para comparar el intercambio de tramas cuando se conecta el FT120 utilizando la placa de evaluación del fabricante (UMFT12XEV) y cuando se usa nuestro diseño hardware. Si los resultados de ambas pruebas no coinciden, probablemente indique que el diseño del sistema implementado no se ha realizado correctamente.

A continuación se muestra una captura del USBlyzer, en la cual podemos ver las diferentes transferencias que se llevan a cabo entre el *host* y el dispositivo.

Type	Seq	Time	Elapsed	Duration	Request	Request Details	Raw Data	I/O	C:I:E	Device Obj...
URB	0008	17:44:12.247	2.57197...		Bulk or Interrupt Transfer	1 byte buffer		in	23:00:81	FFFFE0010...
URB	0011	17:44:12.247	2.57198...		Bulk or Interrupt Transfer	1 byte buffer		in	23:00:81	FFFFE0010...
URB	0028-0011	17:44:12.298	2.62218...	50.207 ...	Bulk or Interrupt Transfer	1 byte data	02	in	23:00:81	FFFFE0010...
URB	0029-0008	17:44:12.298	2.62219...	50.212 ...	Bulk or Interrupt Transfer	1 byte data	02	in	23:00:81	FFFFE0010...
URB	0038	17:44:12.298	2.62224...		Bulk or Interrupt Transfer	1 byte buffer		in	23:00:81	FFFFE0010...
URB	0039	17:44:12.298	2.62224...		Bulk or Interrupt Transfer	1 byte buffer		in	23:00:81	FFFFE0010...
URB	0044	17:44:12.308	2.63272...		Bulk or Interrupt Transfer	1 byte buffer		in	01:00:81	FFFFE0010...
URB	0045	17:44:12.308	2.63272...		Bulk or Interrupt Transfer	1 byte buffer		in	01:00:81	FFFFE0010...
URB	0046-0045	17:44:12.310	2.63550...	2.774 ms	Bulk or Interrupt Transfer	1 byte data	04	in	01:00:81	FFFFE0010...
URB	0047-0044	17:44:12.310	2.63550...	2.783 ms	Bulk or Interrupt Transfer	1 byte data	04	in	01:00:81	FFFFE0010...
URB	0056	17:44:12.311	2.63590...		Bulk or Interrupt Transfer	1 byte buffer		in	01:00:81	FFFFE0010...
URB	0057	17:44:12.311	2.63590...		Bulk or Interrupt Transfer	1 byte buffer		in	01:00:81	FFFFE0010...
URB	0066-0057	17:44:12.427	2.75150...	115.598...	Bulk or Interrupt Transfer	1 byte data	04	in	01:00:81	FFFFE0010...
URB	0067-0056	17:44:12.427	2.75150...	115.605...	Bulk or Interrupt Transfer	1 byte data	04	in	01:00:81	FFFFE0010...
URB	0076	17:44:12.427	2.75191...		Bulk or Interrupt Transfer	1 byte buffer		in	01:00:81	FFFFE0010...
URB	0077	17:44:12.427	2.75192...		Bulk or Interrupt Transfer	1 byte buffer		in	01:00:81	FFFFE0010...
URB	0086-0077	17:44:12.455	2.77950...	27.580 ...	Bulk or Interrupt Transfer	1 byte data	04	in	01:00:81	FFFFE0010...
URB	0087-0076	17:44:12.455	2.77950...	27.588 ...	Bulk or Interrupt Transfer	1 byte data	04	in	01:00:81	FFFFE0010...
URB	0096	17:44:12.455	2.77992...		Bulk or Interrupt Transfer	1 byte buffer		in	01:00:81	FFFFE0010...
URB	0097	17:44:12.455	2.77993...		Bulk or Interrupt Transfer	1 byte buffer		in	01:00:81	FFFFE0010...
PnP	0122	17:44:12.489	2.81466...		Query Interface	PnP Location				FFFFE0010...
PnP	0123	17:44:12.489	2.81466...		Query Interface	PnP Location				FFFFE0010...
PnP	0124	17:44:12.489	2.81466...		Query Interface	PnP Location				FFFFE0010...
PnP	0125	17:44:12.489	2.81466...		Query Interface	PnP Location				FFFFE0010...
PnP	0126-0125	17:44:12.489	2.81467...	3 us	Query Interface	PnP Location				FFFFE0010...
PnP	0127-0124	17:44:12.489	2.81467...	7 us	Query Interface	PnP Location				FFFFE0010...
PnP	0128-0123	17:44:12.489	2.81467...	9 us	Query Interface	PnP Location				FFFFE0010...
PnP	0129-0122	17:44:12.489	2.81467...	11 us	Query Interface	PnP Location				FFFFE0010...
URB	0130	17:44:12.530	2.85581...		Get Descriptor from Device	Dvc		in	--:--:00	FFFFE0010...
URB	0131	17:44:12.530	2.85581...		Get Descriptor from Device	Dvc		in	--:--:00	FFFFE0010...

Figura 4.2.2 Captura del intercambio de tramas con el USBlyzer

El resultado obtenido tras comparar las transferencias en los dos escenarios analizados, es que todas las tramas hasta la solicitud *Get Descriptor* coinciden. Por lo tanto el intercambio de tramas observado utilizando el sistema desarrollado, sería idéntico al de la figura 4.2.2 con la diferencia de que no se llegan a realizar las transacciones desde la solicitud *Get Descriptor*.

4.2.2 Placa principal UMFT12XEV

La placa principal UMFT12XEV, está diseñada por el fabricante FTDI para facilitar el uso del dispositivo FT120. El elemento fundamental es el microcontrolador LPC1114, que está programado para gestionar la comunicación USB entre el PC y el dispositivo FT120, que concuerda con lo desarrollado en nuestro sistema. En definitiva, esta placa proporciona al FT120 lo mismo que nuestro diseño, pero tratándose de una implementación software.

La prueba que se ha realizado consiste en la conexión del dispositivo FT120 a la placa UMFT12XEV, y estos a su vez al ordenador mediante la conexión USB que presenta el dispositivo. En la siguiente imagen se muestra una foto del montaje utilizado.

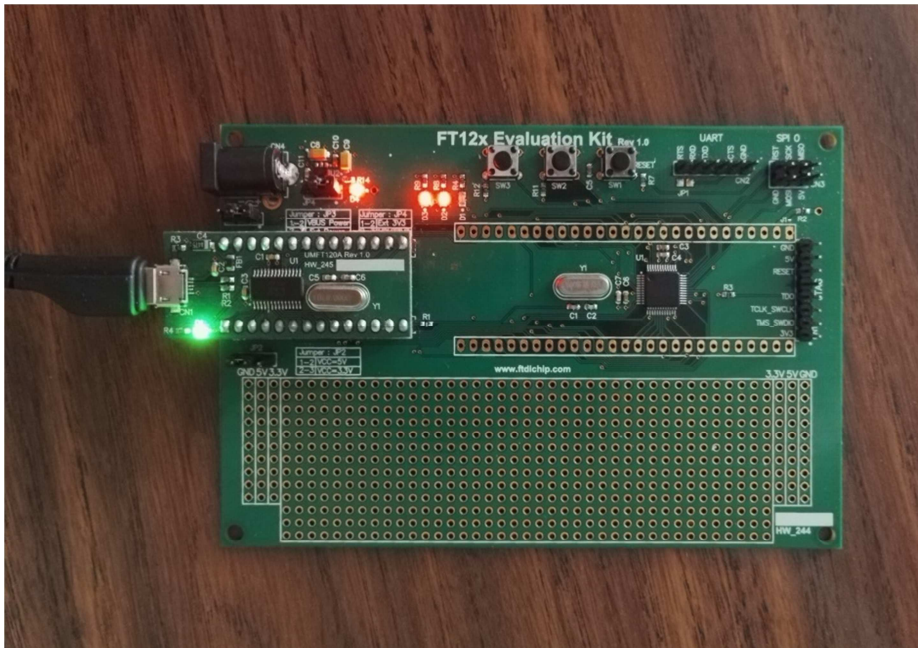


Figura 4.2.3 Montaje del FT120 y la placa UMFT12XEV

Para comprobar si el FT120 está funcionando correctamente, se debe prestar atención al led verde que presenta el dispositivo. Este parpadea cuando hay transacciones USB y permanece fijo cuando se ha completado el proceso de enumeración.

El motivo por el que se utilizó la placa del fabricante inicialmente, fue para comprobar que el ordenador era capaz de establecer una comunicación con el dispositivo. Tras conseguir que se estableciera dicha comunicación instalando en el ordenador los drivers indicados por el fabricante, se procedió posteriormente a conectar el dispositivo en el sistema desarrollado. Tras conectar las resistencias, las tierras y las alimentaciones, se logró que el ordenador detectara el dispositivo y se estableciera una comunicación entre ellos.

Durante el desarrollo del sistema se observó que sucesivas ejecuciones del protocolo, sin realizar cambios entre ellas, proporcionaban resultados diferentes. Unas veces se realizaban correctamente los pasos del proceso de enumeración implementados, mientras que otras, se interrumpían al recibir del dispositivo un reseteo del bus o la suspensión de la comunicación.

Como consecuencia de las anomalías encontradas a la hora de usar el dispositivo, se procedió a realizar la prueba con la placa proporcionada por el fabricante UMFT12XEV. Tras realizar varios intentos, se observa que aunque algunas veces el proceso de enumeración se completa, otras no se consigue y el led indicativo no llega a permanecer fijo. Esto indica que con la placa del fabricante también ocurren fallos en la comunicación, por lo que se asume que también se den en este sistema implementado.

Otro de los motivos por los que se utilizó esta placa, fue para tenerla de referencia a la hora de establecer la conexión entre el ordenador y el dispositivo. Se tiene acceso al código C que programa el microcontrolador y además se observa mediante el USBlyzer el intercambio de tramas entre el *host* USB y el FT120. Las transferencias observadas con el analizador USBlyzer, cuando se utiliza el montaje indicado anteriormente, servirán para compararlas con las obtenidas del sistema desarrollado.

4.2.3 Sistema desarrollado

El montaje que se ha llevado a cabo para la implementación de este proyecto consiste en la conexión del dispositivo FT120 a la plataforma FPGA. Para ello se utiliza la placa de interfaz de módulos FX2 (FX2 MIB), que permite la conexión de cada uno de los pines del FT120 con las entradas/salidas de propósito general (GPIO) de la FPGA. Este sistema a su vez se conecta al ordenador mediante la conexión USB que presenta el dispositivo. En la siguiente imagen se muestra el montaje del sistema desarrollado en este proyecto.

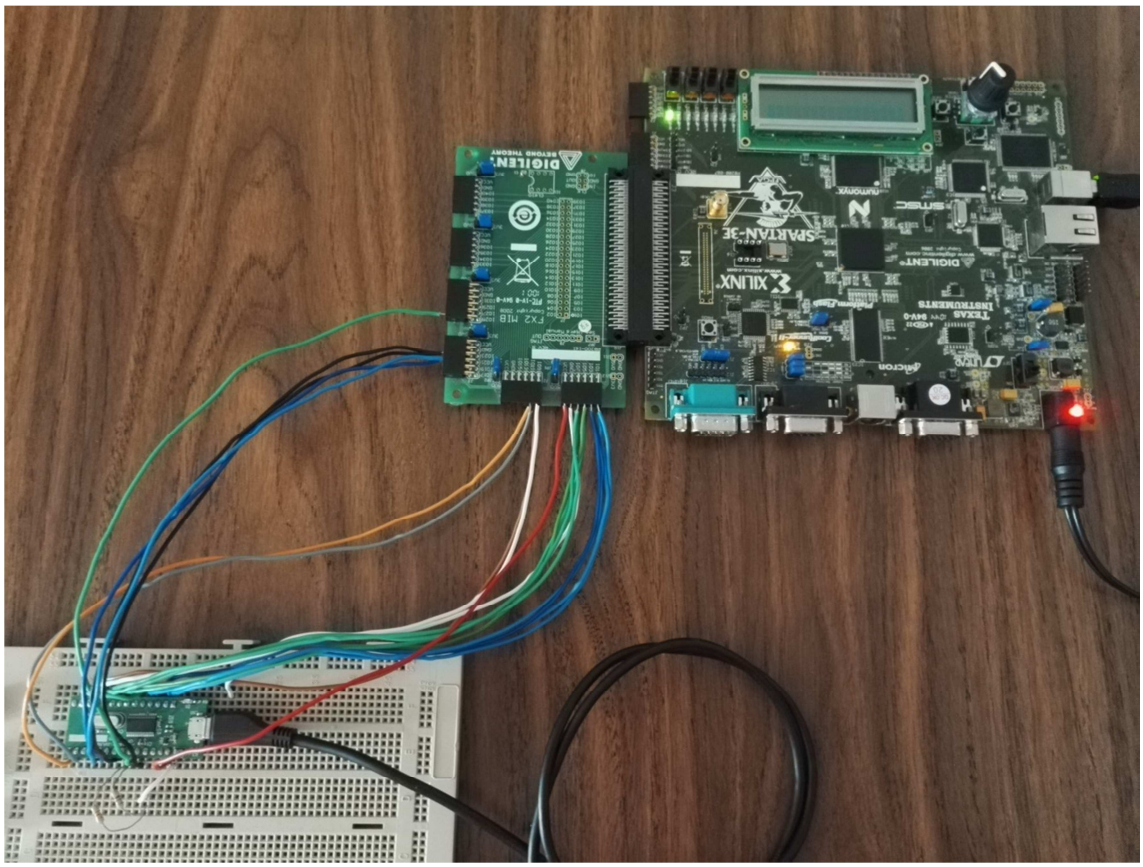


Figura 4.2.4 Montaje del sistema desarrollado

La prueba que se ha realizado consiste en la conexión USB del sistema mostrado anteriormente y la posterior comprobación de su funcionamiento. Para ello, primero se observará el comportamiento que presenta el led del dispositivo y la detección que el ordenador hace de este. Después se procederá a utilizar todos los mecanismos necesarios para depurar y comprobar cómo se comporta el sistema implementado.

Lo primero que se puede comprobar al realizar la conexión USB, es que el sistema es capaz de realizar transferencias, ya que parpadea el led indicativo del FT120. No llega a completarse el proceso de conexión, ya que no permanece fijo el led, pero corresponde con lo esperado al no haberse completado la implementación del proceso de enumeración. Esto a su vez concuerda con lo observado en el administrador de dispositivos del ordenador, donde se confirma que la conexión no se ha completado con éxito. Por otro lado, se corrobora que los drivers han sido instalados correctamente al detectarse la conexión del dispositivo.

A continuación, se procede a realizar la depuración del sistema implementado utilizando diversos mecanismos. Esto permitirá comprobar su funcionamiento e intentar solucionar los problemas que se hayan podido detectar.

Leds y switches

Como se ha explicado anteriormente, se utiliza este mecanismo para observar los valores de las diferentes señales, y así saber el estado en el que se encuentra el sistema y la información que pueda resultar más relevante. Ha sido de mucha utilidad durante todo el proceso de implementación del sistema, siendo uno de los mecanismos principales a la hora de realizar la depuración.

Uno de los principales problemas que fueron detectados mediante la utilización de este mecanismo, fue que ciertas modificaciones en el código implementado provocaban cambios no esperados en la evolución de la comunicación. Se observaba que el estado en el que finalizaba la ejecución iba variando con cierta aleatoriedad al realizar cambios que a priori no deberían afectar al circuito. Estas variaciones inesperadas se atribuyeron a problemas en la temporización, por lo que se probó a ampliar los márgenes temporales del diseño y a modificar en varias ocasiones las escrituras y lecturas realizadas en el sistema.

El resultado obtenido al final del desarrollo de este proyecto, es que la variable de depuración indica que se ha completado el envío del descriptor de dispositivo solicitado. El *flag1q1* se encuentra activado indicando que se ha completado la primera escritura del *buffer*. La señal *wCountq1* vale 18, indicando cuantos datos se han enviado, que corresponde al tamaño del descriptor de dispositivo. La señal *PayloadPacketRq1* contiene los datos obtenidos en la última lectura del *buffer*, que se corresponde con el paquete *setup* de la tabla 3.2.1, que contiene la solicitud del descriptor de dispositivo. Todos estos valores observados corroboran que el sistema implementado funciona correctamente.

Por último, se debe explicar que tras el último paso implementado del proceso de enumeración, es decir, atender la solicitud *Get Descriptor*, el sistema no recibe una nueva solicitud. El sistema llega al estado de espera interrupción y no se vuelve a activar la señal de interrupción, que permitiría al sistema realizar una nueva recepción. El motivo por el que no

continúa el proceso de enumeración no se ha conseguido determinar, por lo que debería ser el punto de partida para continuar con el desarrollo de este sistema.

Analizador lógico

Se ha utilizado un analizador lógico que permite observar el valor de las señales digitales de cada uno de los pines del FT120, es decir, la comunicación que se establece entre el sistema desarrollado en la FPGA y el dispositivo.

No se han conseguido obtener los resultados esperados de este mecanismo debido a las dificultades a la hora de manipular el dispositivo FT120. El motivo es que las soldaduras del conector USB son muy frágiles y se rompieron en varias ocasiones. Se tuvo que recurrir a los técnicos de laboratorio para su reparación, y a la utilización de varios de estos dispositivos. En algunos de ellos se llegaron a romper las pistas del circuito dejando el dispositivo inservible.

Analizador de tiempos

Se ha utilizado la herramienta analizador de tiempos que proporciona el programa Xilinx-ISE. Este permite comprobar el retraso máximo que presenta el circuito desarrollado y si está dentro de los márgenes que permite la frecuencia de funcionamiento utilizada.

Los resultados obtenidos al analizar el sistema implementado son que el retraso máximo es 12,068 ns y el periodo mínimo de funcionamiento es 12,075 ns. Como el periodo de reloj utilizado en este diseño es 20 ns, se cumplen de sobra las restricciones temporales del circuito.

Simulación Post-Synthesis

Se ha realizado una simulación *post-synthesis* del sistema implementado para comprobar su funcionamiento y localizar sus posibles fallos. La generación del correspondiente modelo de simulación se ha llevado a cabo mediante el programa Xilinx-ISE, y la posterior simulación se ha realizado con el programa ModelSim. Para esta simulación se ha utilizado el banco de pruebas explicado en el apartado 4.1.7 Simulación global y la biblioteca *unisim.*

Uno de los motivos por los que se utilizó este mecanismo de comprobación fue detectar el origen del comportamiento aleatorio del circuito explicado en el apartado anterior leds y *switches*. Los resultados obtenidos en esta simulación concuerdan con lo obtenido en las realizadas anteriormente, por lo que no se ha conseguido detectar ningún fallo que explique las anomalías detectadas.

Simulación Post-Place & Route

Se ha realizado una simulación *post-place & route* del sistema para comprobar si se detecta algún problema en el diseño implementado. Al igual que en el caso anterior, se han utilizado los programas Xilinx-ISE, para generar el modelo de simulación *post-place & route,* y ModelSim, para llevar a cabo la simulación. El banco de pruebas utilizado volverá a ser el mismo, pero ahora se necesita la biblioteca *simprim.*

En la simulación *post-place & route* también funciona todo el sistema correctamente, por lo que no nos proporciona indicios de ningún fallo que se pudiera dar en una ejecución real.

USBlyzer

El uso del USBlyzer tiene como objetivo hacer una comparación entre las transferencias que se realizan para cada uno de los montajes que se han detallado, tal y como se ha mencionado anteriormente.

El resultado obtenido es que coinciden las transacciones con las del primer montaje explicado hasta la solicitud *Get Descriptor*, donde se detiene la comunicación. La falta de la trasferencia de la petición *Get Descriptor*, podría ser el motivo por el que tras atender esta solicitud, no se recibe una nueva petición.

No se ha conseguido encontrar el motivo por el que esta solicitud no aparece en el tráfico de datos detectado, ya que el dispositivo si realiza esta petición al sistema desarrollado. Esta cuestión se encuentra relacionada con una de las dificultades que hemos encontrado durante la realización de este proyecto, el proceso que sigue el controlador FT120 para relacionar la comunicación que establece con el *host* USB y la establecida con este sistema.

4.3 Dificultades Encontradas y posibles soluciones

En este apartado se proceden a explicar las principales dificultades encontradas durante la realización de este proyecto. Esto es lo que no ha permitido implementar más partes del protocolo USB y completar todos los objetivos fijados en un primer momento.

Uno de los principales problemas encontrados y que ha dificultado mucho la realización de este proyecto ha sido el comportamiento inestable del dispositivo FT120. En algunas ocasiones se interrumpe la comunicación establecida sin ningún motivo aparente. Esto provocó retrasos en el desarrollo del sistema, debido a la dificultad a la hora de realizar pruebas y comprobar su correcto funcionamiento. Como se ha mencionado en el apartado 4.2.2, se comprobó que este problema también surgía al usar la placa del fabricante, por lo que se asumió que no era un problema del sistema desarrollado.

Otra de las dificultades que se han detectado en el sistema desarrollado, es la aleatoriedad en el comportamiento del circuito, tal y como se ha explicado en el aparatado *leds y switches* del punto 4.2.3. Para encontrar el motivo y proceder a solucionarlo, se realizó una simulación *post-synthesis* y otra *post-place and route*, pero tal y como se ha explicado anteriormente no se detectó este mismo problema en las simulaciones realizadas.

La especificación del dispositivo FT120 indica los comandos que este puede recibir, pero sin detallar cuando se utiliza concretamente cada uno de ellos ni en qué orden. No se dispone de información acerca de cómo el dispositivo actúa ante la recepción de cada una de las tramas enviadas por el *host* USB. Este es otro de los motivos por los que la implementación del protocolo ha costado más de lo esperado. Debido a esto, se ha tenido que analizar cada uno de los datos recibidos por el dispositivo y deducir los siguientes comandos y datos a enviar. Para ello se ha tenido que recurrir en ciertas ocasiones al código C que programa el microcontrolador LPC1114 de la placa UMFT12XEV, ya que en este se implementan los comandos del FT120 que permiten la comunicación USB.

También se deben recalcar las dificultades encontradas para observar que el sistema funciona tal y como corresponde en cada momento. Mediante el programa USBlyzer se han consultado las transacciones entre el PC y el dispositivo, pero debido a la falta de información sobre la correspondencia con los comandos utilizados del FT120, no se puede concluir en que momento de la ejecución se encuentra el sistema implementado. Aunque se han usado los métodos indicados en el punto anterior para realizar las comprobaciones que se han considerado oportunas, no se ha podido realizar una observación concreta de la comunicación entre el dispositivo y el sistema implementado.

El uso del analizador lógico es una de las soluciones que se han planteado para poder comprobar de manera exhaustiva y sin lugar a dudas el estado de la comunicación. El motivo por el que no se han obtenidos resultados es la dificultad encontrada a la hora de manipular el dispositivo FT120, tal y como ya se ha mencionado. Una de las posibles soluciones para poder realizar esta comprobación, sería implementar una PCB más robusta que nos permita la fácil manipulación del dispositivo. Debemos tener en cuenta que la fragilidad de las conexiones del circuito también han provocado fallos en las pruebas realizadas, ya que en ocasiones se ha podido ignorar que el dispositivo se encontraba dañado, provocando un gran retraso.

5. Conclusiones

En primer lugar, se debe resaltar que los objetivos intermedios fijados y fundamentales para la realización de las transacciones entre el dispositivo FT120 y el *host* USB, se han logrado implementar correctamente. Estos objetivos son enviar comando, enviar dato, recibir dato, escribir *buffer* y leer *buffer*.

Por otro lado, los objetivos fijados durante el desarrollo de este proyecto se han conseguido sólo de manera parcial. El objetivo fijado en un primer momento era completar el proceso de enumeración del dispositivo USB, con el que se conseguiría establecer la conexión. Se han completado solo una parte de los pasos del proceso de enumeración del dispositivo USB utilizado.

En este proyecto se han llevado a cabo la inicialización y configuración del chip, el manejo de interrupciones del FT120 y los *endpoints* de control, la recepción de los paquetes de *Setup*, la atención de la solicitud *Get Descriptor* y el envío del resto de datos pendientes. Esto corresponde a la implementación de los 8 primeros pasos del proceso de enumeración, de los 12 que son en total.

Además, se debe tener en cuenta que con los pasos implementados, se tiene la base fundamental para la realización del sistema completo en un futuro. Para completar el resto de pasos del proceso de enumeración, se puede reutilizar gran parte del diseño realizado, ya que el procedimiento para atender las diferentes peticiones del *host* USB es el mismo, cambiando la información enviada por la que corresponda.

Los motivos por los que no se ha conseguido completar el objetivo fijado, han sido las dificultades encontradas durante el desarrollo de este proyecto. Algunas de las más importantes son: fallos en la comunicación, variaciones inesperadas, soldaduras débiles, falta de información sobre el uso de los comandos y de las tramas correspondientes.

A parte de los resultados obtenidos, se ha de tener en cuenta que ha habido un gran trabajo previo a la realización del sistema implementado. Se ha realizado un estudio detallado del protocolo USB, el cual se desconocía al comienzo de este proyecto. Además, se ha investigado todo lo necesario para poder utilizar el dispositivo FT120 en nuestra implementación, ya que no se había trabajado con él con anterioridad.

Por último, se debe mencionar todo lo aprendido durante la realización de este proyecto. Primeramente, el funcionamiento del protocolo USB, que aun siendo algo que se utiliza en la actualidad con mucha frecuencia, se desconocía al comienzo del mismo. Por otro lado, la utilización de dispositivos de la familia FTDI, en concreto el FT120. También se han ampliado los conocimientos del lenguaje VHDL y de las herramientas Xilinx-ISE y ModelSim, mediante los cuales se ha desarrollado el sistema implementado en este proyecto.

6. Trabajos Futuros

En primer lugar, se debería detectar el motivo por el que no se recibe una nueva solicitud tras atender la solicitud *Get Descriptor*. Para ello se podría empezar por comprobar si se produce algún error en la comunicación entre el *host* y el FT120, o entre el FT120 y el sistema que se ha implementado (FPGA). Se recomendaría el diseño de una PCB más robusta para el FT120, que permitiera su manipulación y así poder utilizar un analizador lógico para comprobar su funcionamiento.

Una vez solucionado el problema anterior, se debería de completar el proceso de enumeración, implementando los pasos que se han quedado pendientes. Estos pasos serían: atender la solicitud que asigna una dirección (*Set Address*), proporcionar el resto de descriptores cuando se solicitan y atender la solicitud de selección de configuración (*Set Configuration*). Tras completarse este proceso, se podrá comprobar la correcta conexión del dispositivo observando si el led verde del FT120 permanece encendido.

Otro de los trabajos futuros que se podrían realizar, es una mejora en el montaje desarrollado, que se muestra en la figura 4.2.4. Este cambio permitiría prevenir fuentes de ruido que pudieran afectar al funcionamiento del sistema implementado.

El siguiente objetivo sería implementar el envío y la recepción de datos, que proporcionarán la funcionalidad básica de la interfaz USB. Cuando se compruebe que el sistema realiza las transacciones correctamente, se podrá decir que se ha diseñado una interfaz USB sencilla.

Después de implementar la transferencia de datos, se deberá comprobar si se cumplen los objetivos de velocidad fijados. Se deberá verificar que la comunicación es full speed y por lo tanto que la tasa de envío de información es de 12 Mbps. Esto significará que la nueva interfaz USB diseñada mejorará la velocidad que proporcionaban los puertos serie que tenía la FPGA inicialmente, por ejemplo los 20 kbps de la RS-232.

7. Fases del proyecto

La realización de este proyecto se ha dividido en las siguientes fases, que se enumeran y explican a continuación.

1. **Estudio del protocolo USB.** Esta fase es fundamental antes de comenzar con el desarrollo del proyecto, ya que es necesario comprender el funcionamiento del protocolo USB al detalle para poder implementar la interfaz USB, que es el objetivo fundamental.

2. **Estudio del dispositivo FT120.** Para la realización de este proyecto se utiliza el dispositivo FT120, que permite la comunicación entre el puerto USB del ordenador y la entrada/salida de propósito general de la FPGA. Por este motivo es necesario el estudio de todas las especificaciones y documentos encontrados sobre dicho dispositivo.

3. **Conexión del FT120 al PC.** En esta fase se debe establecer la conexión del dispositivo FT120 con el ordenador. Para ello se necesitan instalar en el ordenador los drivers adecuados, permitiendo la comunicación entre ambos.

4. **Conexión entre la FPGA y el FT120.** El siguiente paso es conectar los pines de entrada/salida del dispositivo a las entradas/salidas de propósito general de la FPGA, asignando las señales que correspondan a cada una de ellas. Además se añadirán las alimentaciones y resistencias necesarias, tal y como se indica en la especificación del dispositivo.

5. **Inicialización y configuración.** En este apartado se inicializa y configura el FT120, para lo que se debe realizar el primer envío de comandos y datos.

6. **Manejador de interrupciones.** Una vez completada la fase anterior, se implementa la atención a las interrupciones recibidas del FT120. Para ello se debe realizar la primera lectura de datos, que corresponderá al valor del registro de interrupción.

7. **Bloqueo de los *endpoints* de control.** En esta fase se bloquean mediante los comandos del FT120, los *endpoints* de control que se indican en el registro de interrupción. Para ello debemos asegurarnos de que no han sido asignados previamente.

8. **Recepción del paquete *setup*.** El siguiente paso es la recepción del paquete *setup*, para lo que se debe realizar una lectura del *buffer*. Los datos recibidos son analizados y se procederá a atender la solicitud correspondiente.

9. **Atención de la solicitud *Get Descriptor*.** Cuando la solicitud recibida se trata de una petición *Get Descriptor*, se procede al envío del descriptor solicitado, para lo que se realiza la primera escritura en el *buffer*.

10. **Escribir el resto de los datos.** Cuando los datos solicitados mediante el paquete *setup* superan el tamaño máximo de paquete, se realizarán varias escrituras en el *buffer* hasta completar la información solicitada.

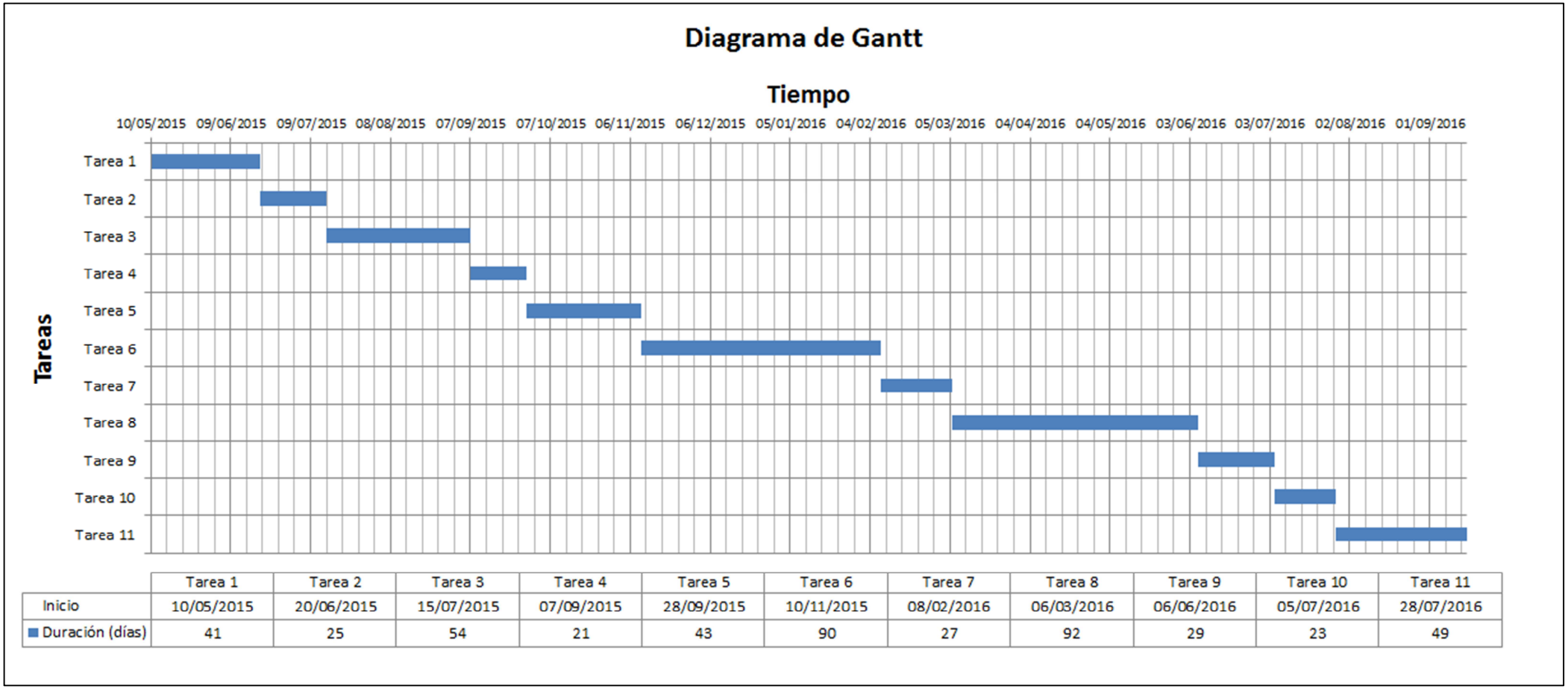
11. **Redacción de la memoria del proyecto.** La última fase de este proyecto consiste en la redacción de esta memoria, que contendrá toda la información necesaria para entender y reproducir el proyecto implementado.

Las distintas fases del proyecto y su correspondiente duración se recogen en la siguiente tabla.

Tareas	Inicio	Fin	Duración (días)
Estudio del protocolo USB	10/05/2015	20/06/2015	41
Estudio del dispositivo FT120	20/06/2015	15/07/2015	25
Conexión del FT120 al PC	15/07/2015	07/09/2015	54
Conexión de la FPGA y el FT120	07/09/2015	28/09/2015	21
Inicialización y configuración	28/09/2015	10/11/2015	43
Manejador de interrupciones	10/11/2015	08/02/2016	90
Bloqueo de los *endpoints* de control	08/02/2016	06/03/2016	27
Recepción del paquete *setup*	06/03/2016	06/06/2016	92
Atención de la solicitud *Get Descriptor*	06/06/2016	05/07/2016	29
Escribir el resto de datos	05/07/2016	28/07/2016	23
Redacción de la memoria del proyecto	28/07/2016	15/09/2016	49
		Total	494

Tabla 7.1.1 Fases del proyecto

7.1 Diagrama de Gantt



	Tarea 1	Tarea 2	Tarea 3	Tarea 4	Tarea 5	Tarea 6	Tarea 7	Tarea 8	Tarea 9	Tarea 10	Tarea 11
Inicio	10/05/2015	20/06/2015	15/07/2015	07/09/2015	28/09/2015	10/11/2015	08/02/2016	06/03/2016	06/06/2016	05/07/2016	28/07/2016
Duración (días)	41	25	54	21	43	90	27	92	29	23	49

Figura 7.1.1 Diagrama de Gantt

8. Presupuesto

En este apartado se detallan todos los costes derivados del proyecto. Para ello se tiene en cuenta el tiempo de desarrollo que se detalla en el aparatado anterior, un total de 494 días que corresponden a 16 meses de trabajo. Los costes de este proyecto se dividen en costes de personal, amortizables y de material.

8.1 Costes de personal

Categoría	Dedicación	Coste mensual	Coste (Euro)
Jefe de proyecto	2 %	4.150,00	1328,00
Ingeniero de desarrollo	50 %	2.490,00	19.920,00
		Total	21.248,00

Tabla 8.1.1 Costes de personal

Para el cálculo del coste de personal se ha tenido en cuenta el porcentaje de tiempo dedicado al proyecto durante la realización del mismo. Además, se ha tenido en cuenta que el coste mensual de un trabajador es un 66% más de su salario mensual. Basándose en todo esto los costes obtenidos son los mostrados en la tabla 8.1.1.

8.2 Costes Amortizables

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación (meses)	Coste imputable
Ordenador portátil	600	100	16	48	200,00
Licencia ModelSim	Gratuita	100	16	-	0,00
Licencia ISE	Gratuita	100	16	-	0,00
Licencia USBlyzer	Gratuita	100	16	-	0,00
				Total	200,00

Tabla 8.2.1 Costes amortizables

Para los costes amortizables se ha calculado la relación entre los meses que se ha utilizado un producto respecto a su periodo de depreciación. Mediante este valor se calcula el coste del producto imputable a este proyecto.

8.3 Costes de material

Material	Unidades	Coste unitario (Euro)	Coste total (Euro)
Spartan-3E Starter Board FPGA	1	266,00	266,00
FX2 MIB	1	26,67	26,67
UMFT120DC	3	9,57	28,71
UMFT12XEV	1	50,64	50,64
Cable y Resistencias	-	2,50	2,50
		Total	374,52

Tabla 8.3.1 Costes de material

Los costes de los materiales utilizados para la realización del proyecto se detallan en la anterior tabla 8.3.1. Se enumeran todos los elementos empleados y el precio de la adquisición de cada uno de ellos.

8.4 Costes totales

Concepto	Presupuesto Costes Totales
Personal	21.248,00
Amortización	200,00
Material	374,52
Costes Indirectos	4364,50
Total	**26.187,02**

Tabla 8.4.1 Costes totales

En la tabla 8.4.1 se muestran los costes asociados a cada uno de los conceptos, los cuales se han calculado en los apartados anteriores. Además, se incluyen los costes indirectos que corresponden a un 20% adicional. La suma de todos ellos nos proporciona el coste total del proyecto.

El presupuesto total de este proyecto asciende a la cantidad de 26.187,02 euros.

9. Glosario

***Buffer*:** Memoria de almacenamiento temporal de información que permite transferir los datos entre unidades funcionales con características de transferencia diferentes.

CRC: La verificación de redundancia cíclica, es un método de control de integridad de datos de fácil implementación. Es el principal método de detección de errores utilizado en las telecomunicaciones.

DMA: El acceso directo a memoria, es una característica de las computadoras y microprocesadores modernos que permite que ciertos subsistemas de hardware dentro de la computadora puedan acceder a la memoria del sistema para la lectura y/o escritura, independientemente de la unidad central de procesamiento (CPU).

DPLL (*Digital Phase-Locked Loop*): Sistema de control que genera una señal de salida cuya fase está relacionada con la fase de una señal de entrada. En este caso el detector de fase es digital.

***Endpoint*:** Un *endpoint* es una porción única direccionable de un dispositivo USB que es origen o destino de información en un flujo de comunicación entre el *host* y el dispositivo.

Flag: En programación, la bandera o *flag* se refiere a uno o más bits que se utilizan para almacenar un valor binario o código que tiene asignado un significado

FPGA (*Field Programmable Gate Array*): Una FPGA es un dispositivo lógico programable, es decir, un chip cuyas puertas lógicas a nivel físico podemos programar.

FSM *(Finite State Machine)***:** Una máquina de estados es una estructura de programa que nos sirve para determinar el comportamiento de algo en base al estado en el que se encuentre. Para cada estado por tanto se tendrá un comportamiento.

Full-speed: La tasa de velocidad completa (*full-speed*) es de 12 Mbit/s, corresponde con la velocidad de datos USB básica definida por el USB 1.0. Todos los *hubs* USB pueden funcionar a esta velocidad.

Half-duplex: Circuito o canal de comunicaciones que puede transmitir información alternativamente, pero no de forma simultánea, en dos direcciones.

Handshake (Apretón de Manos): Protocolo de comienzo de comunicación entre dos máquinas o sistemas.

High-speed: La tasa de alta velocidad (*high-speed*) es de 480 Mbit/s. Los dispositivos de alta velocidad son compatibles con los USB 1.1 y se definen en el USB 2.0.

***Host*:** El host (anfitrión) es un nodo, un ordenador o un conjunto de ellos, que ofrece servicios y datos a los dispositivos conectados a él. Dentro del sistema USB existe un único host, el cual inicia la comunicación.

***Hub*:** Dispositivo que contiene uno o más conectores o conexiones internas hacia otros dispositivos usb, el cual habilita la comunicación entre el *host* y con diversos dispositivos. Cada conector representa un puerto USB.

Low-speed: La tasa de baja velocidad es de 1.5 Mbit/s y viene definida por el USB 1.0.

NRZI: Esta es la codificación no retorno a cero invertido, que sirve para pasar de señales binarias a señales físicas. Se producen transiciones cuando la señal binaria vale 1 y no se producen cuando vale 0.

Pipe: Una pipe es un enlace virtual entre un *endpoint* de un dispositivo y el software del *host*, que permite mover los datos de uno a otro a través de un *buffer* de memoria.

Reset: Este término equivale al reinicio en español, y significa volver a las condiciones iniciales de un sistema.

VHDL (*VHSIC Hardware Description Language*): Lenguaje para describir circuitos integrados, reduciendo los tiempos de diseño de los circuitos.

VHSIC (*Very High Speed Integrated Circuit*): Como su nombre indica se refiere a un circuito integrado de muy alta velocidad.

USB (*Universal Serial Bus*): Un puerto USB funciona como dispositivo que facilita la conexión de periféricos y accesorios a un ordenador, permitiendo el fácil intercambio de datos y la ejecución de operaciones.

10. Referencias

[1] González, M. y Portela, M., Estudio Tecnológico: Elección de Controlador USB, Departamento de Tecnología Electrónica, Universidad Carlos III de Madrid, Junio 2014

[2] Universal Serial Bus Specification, Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips, Revision 2.0, April 27, 2000

[3] Cypress Semiconductor Corporation, Murphy, R., USB 101: An Introduction to Universal Serial Bus 2.0, AN57294, 2009-2016

[4] Axelson, J.: USB Complete, second edition, Lakeview Research, Madison, WI, 2001

[5] Future Technology Devices International Ltd, FT120 USB Device Controller with Parallel Bus IC, Datasheet Version 1.0, 2012

[6] Future Technology Devices International Ltd, FT12 Series Evaluation Kit, Datasheet Version 1.0, 2012

[7] Xilinx, Spartan-3E FPGA Family Data Sheet, DS312, Product Specification, July 2013

[8] Xilinx, Spartan-3E Starter Kit Board User Guide, UG230 (v1.0) March 9, 2006

[9] López Vallejo, M. L. y Ayala Rodrigo, J. L., FPGA: Nociones básicas e implementación, Departamento de Ingeniería Electrónica, Universidad Politécnica de Madrid, Abril 2004

[10] Future Technology Devices International, FT12 Series Firmware Programming Guide, Application Note AN_225, Version 1.0, September 2012

Anexo 1. Diagrama de flujo Principal

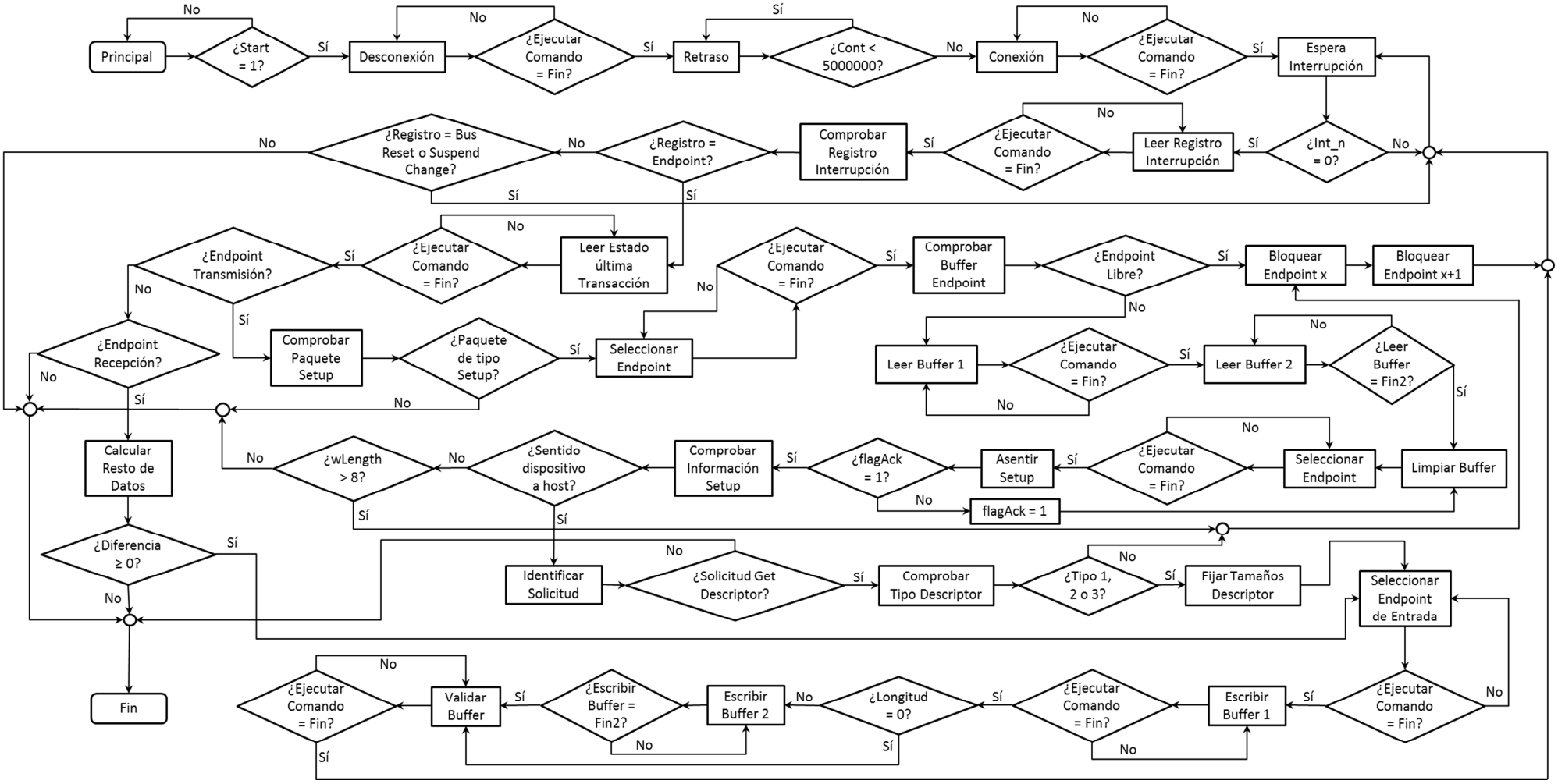


Figura 3.2.2 Diagrama de flujo de la máquina de estados Principal